НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«На правах рукопису»
УДК _____

До захисту допущено:
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Інженерія програмного забезпечення комп'ютеризованих систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Розподілене машинне навчання з використанням технології Apache Spark»

Виконав (-ла):
студент (-ка) VI курсу, групи ІП-81мн
Мірошник Олексій Сергійович _____

Керівник:
Старший викладач
Олійник Юрій Олександрович _____

Рецензент:
Доц. каф. ОТ, к.т.н.
Волокита Артем _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Автоматизованих систем обробки інформації і управління

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма - «Інженерія програмного забезпечення комп'ютеризованих систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Мірошнику Олексію Сергійовичу

1. Тема дисертації «*Розподілене машинне навчання з використанням технології Apache Spark*», науковий керівник дисертації Олійник Юрій Олександрович, старший викладач, затверджені наказом по університету від «24» березня 2020 р. №910-с

2. Термін подання студентом дисертації *27.04.2020*

3. Об'єкт дослідження *Процеси розподіленого машинного навчання*

4. Предмет дослідження *Методи розподіленого машинного навчання*

5. Перелік завдань, які потрібно розробити *Проаналізувати існуючі методи та підходи до розподіленого машинного навчання; зібрати навчальні дані та сформувати набори для виконання розподілення; розробити метод розподіленого машинного навчання на прикладі алгоритму дерев ізоляцій; протестувати та проаналізувати ефективність отриманого методу*

6. Орієнтовний перелік графічного (ілюстративного) матеріалу *інформаційні потоки процесу машинного навчання; схема машинного навчання з використанням потокової обробки даних*

7. Орієнтовний перелік публікацій *Задача розподіленого машинного навчання, Розподілене машинне навчання з використанням технології Apache Spark*

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	*Огляд існуючих підходів до розподіленого машинного навчання*	*30.11.2018*	
2	*Постановка та формалізація математичної моделі задачі*	*15.03. 2019*	
3	*Розробка методу розподіленого навчання*	*10.10.2019*	
4	*Проведення експериментальних досліджень розробленого алгоритму*	*20.12.2019*	
5	*Оформлення документації*	*19.03.2020*	
6	*Подання дисертації на попередній захист*	*23.04.2020*	
7	*Подання дисертації на основний захист*	*19.05.2020*	

Студент Олексій Мірошник

Науковий керівник Юрій Олійник

РЕФЕРАТ

Магістерська дисертація: 102 с., 40 рис., 1 табл., 2 додатки, 62 джерела.

Актуальність теми: засоби машинного навчання використовується там, де звичайні алгоритми не можуть бути застосовані через складність задачі та неможливість вирішення її традиційними методами. Проте, об’єми даних необхідні для навчання невпинно ростуть і все частіше не можуть бути швидко та ефективно опрацьовані одним робочим пристроєм. Вирішенням цієї проблеми є використання розподілених обчислень та застосування таких підходів до задач машинного навчання з використанням розподілених систем з декількома обчислювальними вузлами та мережевою взаємодією між ними. За рахунок розподілення можна не лише пришвидшити навчання, а й збільшити пропускну здатність, використовувати потоки даних, виконувати оптимізації над моделями, навчати різні версії паралельно тощо.

Мета дослідження: прискорення машинного навчання за рахунок методу розподіленого машинного навчання на прикладі вирішення задачі пошуку аномалій з використанням дерев ізоляцій.

Для реалізації поставленої мети були сформульовані **наступні завдання**:

– виконати аналіз існуючих методів та підходів до розподіленого машинного навчання;

– збір навчальних даних та формування наборів для виконання розподілення;

– розробити метод розподіленого машинного навчання на прикладі алгоритму дерев ізоляцій;

– тестування та аналіз ефективності отриманого методу;

– визначення подальшого напрямку досліджень.

Об’єкт дослідження: процеси розподіленого машинного навчання.

Предмет дослідження: методи розподіленого машинного навчання.

Методи дослідження: для розв’язання поставленої задачі використовувались дерева та ліс ізоляцій, розподілені обчислення, файлова система GFS, обчислювальний підхід MapReduce, потоки даних.

Наукова новизна: науковим результатом магістерської дисертації є створення методу розподіленого навчання на основі використання розподілених даних, обчислювальних ресурсів та залучення потокової обробки даних.

Практичне значення отриманих результатів: визначається тим, що запропонований метод дозволяє прискорити навчання моделей з використанням дерев ізоляцій, збільшити відмовостійкість системи, підтримувати прозору масштабованість для користувача.

Зв’язок роботи з науковими програмами, планами, темами: робота виконувалась на кафедрі автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського» в рамках теми «Методи та технології високопродуктивних обчислень та обробки надвеликих масивів даних». Державний реєстраційний номер 0117U000924.

Апробація: основні положення роботи доповідались і обговорювались на IV всеукраїнській науково-практичній конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2020), а також на XVI міжнародній науковій конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту (ISDMCI’2020)» результати магістерської дисертації докладались на наукових конференціях.

Ключові слова: МАШИННЕ НАВЧАННЯ, РОЗПОДІЛЕНЕ НАВЧАННЯ, ПОШУК АНОМАЛІЙ, ПОТОКОВА ОБРОБКА, ПОТОКИ ДАНИХ.

ABSTRACT

Master dissertation: 102 p., 40 fig., 1 tab., 2 sup., 62 sources.

Relevance: machine learning methods are used where conventional algorithms cannot be applied due to the complexity of the problem and the impossibility of solving it by traditional methods. However, the amount of data needed for learning is constantly growing and increasingly cannot be processed quickly and efficiently by a single work device. The solution to this problem is the use of distributed computing and the application of such approaches to machine learning problems using distributed systems with multiple computing nodes and network interaction between them. Distribution can not only speed up learning, but also increase bandwidth, use data streams, perform optimizations on models, teach different versions in parallel, and more.

Purpose: an acceleration of machine learning due to the method of distributed machine learning on the example of solving the problem of finding anomalies using isolation trees.

To achieve this goal, the following tasks were formulated:

– perform an analysis of existing methods and approaches to distributed machine learning;

– collection of training data and formation of sets for distribution;

– to develop a method of distributed machine learning on the example of the isolation tree algorithm;

– testing and analysis of the effectiveness of the obtained method;

– determining the further direction of research.

Object of study: processes of distributed machine learning.

Subject of study: methods of distributed machine learning.

Research methods: isolation forest and trees, distributed computing, GFS file system, MapReduce computational approach, data flows were used to solve this problem.

Scientific novelty: The scientific result of the master's dissertation is the creation of a method of distributed learning based on the use of distributed data, computing resources and the involvement of streaming data processing.

The practical value: is determined by the fact that the proposed method allows to accelerate the learning of models using isolation trees, increase the fault tolerance of the system, and maintain transparent scalability for the user.

Relationship with working with scientific programs, plans, topics: work was performed at the Department of Automated Information Processing and Management Systems of the Igor Sikorsky National Technical University of Ukraine «Kyiv Polytechnic Institute» within the topic «Methods and technologies of high-performance computing and processing of large data sets». State registration number 0117U000924.

Approbation: The main provisions of the work were reported and discussed at the IV All-Ukrainian Scientific and Practical Conference of Young Scientists and Students «Information Systems and Management Technologies» (ISTU-2020), as well as at the XVI International Scientific Conference «Intellectual Systems of Decision-making and Problem of Computational Intelligence» (ISDMCI'2020).

Keywords: MACHINE LEARNING, DISTRIBUTED LEARNING, ANOMALY DETECTION, STREAM PROCESSING, DATA FLOWS.

ЗМІСТ

ВСТУП

Попит на штучний інтелект значно зріс за останнє десятиліття, і це зростання було зумовлене прогресом в техніці машинного навчання та можливістю використання апаратного прискорення.

Однак, щоб підвищити якість прогнозування та зробити рішення машинного навчання можливими для більш складних застосувань, необхідна значна кількість навчальних даних. Незважаючи на те, що невеликі моделі машинного навчання можуть бути навчені з невеликими обсягами даних, вхід для тренінгу великих моделей, таких як нейронні мережі, зростає в експоненціальній залежності від кількості параметрів.

Оскільки попит на обробку даних про навчання випереджав збільшення обчислювальної потужності обчислювальної техніки, існує потреба в розподілі навантажень машинного навчання на декілька машин та перетворенні централізованої в розподілену систему. Ці розподілені системи представляють нові виклики, насамперед ефективну паралелізацію навчального процесу та створення цілісної моделі.

Для досягнення мети необхідно дослідити існуючі підходи та методи до розподіленого машинного навчання, дослідити методи оптимізації використовуваних ресурсів, розглянути варіант з залученням потокової обробки даних до процесу навчання. Необхідно створити метод розподілення машинного навчання на прикладі вирішення задачі пошуку аномалій з використанням дерев ізоляцій.

1 ЗАГАЛЬНИЙ ОГЛЯД МЕТОДІВ ТА ПІДХОДІВ ДО РОЗПОДІЛЕНОГО МАШИННОГО НАВЧАННЯ

Проблема обробки великого обсягу даних у кластері з декількома робочими вузлами не обмежується машинним навчанням, але тривалий час вивчалася в розподілених системах та дослідженнях баз даних. Як результат, деякі практичні впровадження використовують розподілені платформи загального призначення як основу для розподіленого машинного навчання.

Популярні інструменти обробки великих масивів даних, такі як Apache Spark [1], поєднують свої підходи до обробки даних з машинним навчанням, що відображено у спеціалізованих та оптимізованих бібліотеках. Прикладом такої є MLlib [2]. З іншого боку, існує цілий спектр інструментів і засобів створених для машинного навчання, які спочатку були розроблені для роботи на одній машині, а згодом почали отримувати підтримку виконання в розподілених середовищах. Наприклад, відома у світі машинного навчання бібліотека Keras [3] запускається та працює поверх середовища Tensorflow Google [4].

Незважаючи на те, що більшість цих систем призначені для налаштування та експлуатації користувачем локально на одному або декількох пристроях, зростає розмаїття сервісів машинного навчання на базі хмарних рішень. Такі сервіси зосереджені на створені систем розподіленого машинного навчання та надають послуги з розширення функціоналу або збільшення обчислювальних ресурсів.

1.1 Обчислювальна складність машинного навчання

Незважаючи на велике різноманіття алгоритмів та підходів до машинного навчання, представлення даних, що використовуються серед більшості алгоритмів, схожі між собою за структурою. Більшість обчислень у обрахунках машинного навчання складають основні перетворення на векторах, матрицях або тензорах, що є базовими та відомими задачами лінійної алгебри. Необхідність оптимізації таких операцій мали великий

попит та активно обговорювалася у спільнотах з досліджень високопродуктивних обчислень. Як результат одна з таким спільнот, а саме HPC розробила прийоми і бібліотеки BLAS [5] та MPI [6], які були успішно прийняті та інтегровані в системи машинного навчання. У той же час спільнота HPC визначила, що машинне навчання є новим великим навантаженням і почало застосовувати до них методологію HPC.

Як і для інших масштабних обчислювальних завдань, існують два принципово різних та допоміжні способи прискорення робочих навантажень: додавання більшої кількості ресурсів до одного вузла (вертикальне масштабування) та додавання в систему більшої кількості вузлів (горизонтальне масштабування).

1.1.1 Вертикальне масштабування

Серед рішень щодо вертикального масштабування (рисунок 1.1) найпоширенішим методом є додавання програмованих графічних процесорів (англ. *Graphics processing unit, GPU*), і різні систематичні електронні ортези показали переваги цього (наприклад, [7], [8], [9]). Графічні процесори відрізняються високою кількістю апаратних потоків, що робить їх швидшими для глибинного навчання, ніж звичайний серверний процесор. Коли суттєвий ступінь паралелізму регулюється навантаженням, GPU можуть значно прискорити алгоритми машинного навчання.

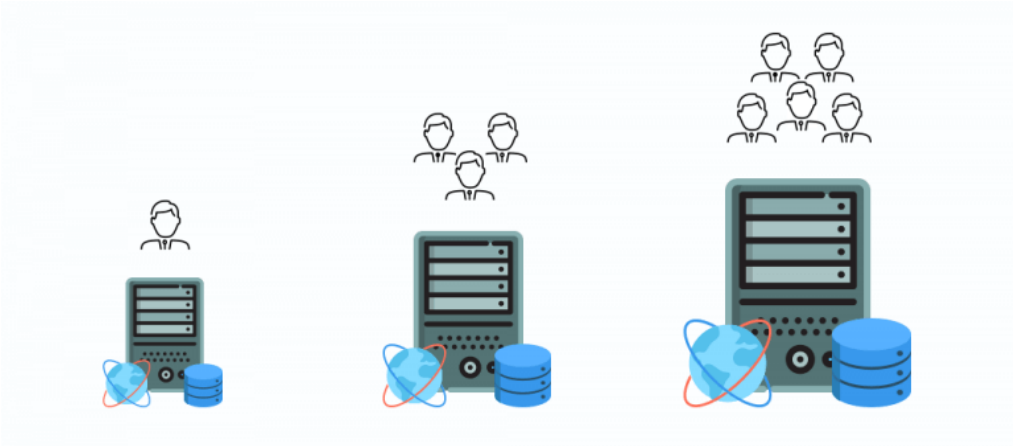

Рисунок 1.1 – Графічне представлення вертикального масштабування

Графічні процесори відрізняються високим рівнем кількість апаратних потоків. Наприклад, Nvidia Titan V і Nvidia Tesla V100 мають 5120 ядер, що робить їх приблизно в 47 разів швидшими для глибинного навчання, ніж звичайний серверний процесор (а саме Intel Xeon E5-2690v4) [10].

Спочатку застосування графічних процесорів для машинного навчання було обмеженим, оскільки графічні процесори використовували чисту SIMD (англ. *single instruction, multiple data* – єдину інструкцію, кілька даних) [11], яка не дозволяла ядрам виконувати іншу гілку коду; всі потоки повинні були виконувати точно таку ж програму. З роками GPU перейшли до більш гнучких архітектур, де накладні розбіжності гілок зменшуються, але розбіжні гілки все ще неефективні[12].

Поширення GPGPU (англ. *General-purpose computing on graphics processing units,* GPU загального призначення, тобто графічних процесорів, які можуть виконувати довільний код) змусило виробників графічних процесорів розроблювати користувацькі пристрої, які можна додавати до звичайних вузлів у якості прискорювачів і більше не виконувати жодної ролі в графічній підсистемі вузла. Наприклад, серія GPU Nvidia Tesla призначена для високопаралельних обчислень і для впровадження в суперкомп'ютери та кластери.

Коли робоче навантаження підлягає достатній мірі паралелізму, GPU можуть значно прискорити алгоритми машинного навчання. Наприклад, Meuth [13] повідомив про збільшення швидкості до 200 разів над звичайними процесорами для алгоритму розпізнавання зображень за допомогою попередньо перевіреного багатошарового персептрону (англ. *Multilayer perceptron, MLP*).

Альтернативою графічним процесорам для прискорення є використання прикладних інтегральних схем (англ. *Application Specific Integrated Circuits,* ASIC), які реалізують спеціалізовані функції завдяки високо оптимізованим конструкціям. Останнім часом попит на такі мікросхеми помітно зростав [14]. Прикладом того є видобуток біткоїнів

(англ. bitcoins). ASIC мають значну конкурентну перевагу перед GPU та процесорами завдяки високій продуктивності та енергоефективності [15]. Оскільки матричне множення відіграє важливу роль у багатьох алгоритмах машинного навчання, ці навантаження добре піддаються прискоренню через ASICS. Google застосував цю концепцію у власне розробленому Tensor Processing Unit (TPU) [16], який, як можна здогадатися з назви, є ASIC, що спеціалізується на обчисленнях тензорів (n-мірних масивів), і призначений для прискорення інструменту Tensorflow [4], який є одним з найпопулярніших будівельних блоків моделей машинного навчання.

Найважливішим компонентом TPU є його одиниця множення матриці на основі систолічного масиву. TPU використовує архітектуру MIMD (англ. *Multiple Instruction stream, Multiple Data stream* – кілька інструкцій, кілька даних) [11], яка, на відміну від GPU, дозволяє їм ефективно виконувати розбіжні гілки. TPU підключаються до серверної системи через шину PCI Express. Це надає їм пряме з'єднання з процесором, що дозволяє забезпечити високу сукупну пропускну здатність 63 Гб/с (PCI-e5x16). У дата-центрах можуть використовуватися декілька TPU, а окремі підрозділи можуть співпрацювати в розподілених налаштуваннях.

Перевага TPU над звичайними налаштуваннями процесора або графічного процесора – це не тільки його більша потужність, але й його енергоефективність, що важливо для великих і масштабних застосувань через витрати енергії та обмежень у дата-центрах. Під час виконання тестів, Jouppi та ін. [17] встановили, що продуктивність TPU може бути більшою у 200 разів, ніж у традиційній системі. Подальше тестування від Sato et al. [16] вказало, що загальна потужність процесора TPU або GPU може бути до 70 разів вищою, ніж центрального процесора для типової нейронної мережі. Підвищення продуктивності варіюється від 3,5 до 71 разів, залежно від завдання, що знаходиться в роботі.

Chen та ін. [18] розробили DianNao, апаратний прискорювач для масштабних нейронних мереж. Їх конструкція впроваджує

нейрофункціональний блок (англ. *Neuro-Functional Unit, NFU*) в потік обробки даних, який помножує всі входи, додає результати і, поетапно, після того, як всі доповнення будуть виконані, необов'язково застосовує функцію активації, як сигмоподібну функцію. Експериментальна оцінка з використанням різних шарів декількох великих нейромережевих структур ([19], [20], [21], [22]) показує прискорення продуктивності на три порядки та зменшення енергії більш ніж 20 разів порівняно з використанням 128-бітового 2 ГГц SIMD-процесора загального призначення.

Навіть процесори загального призначення збільшили доступність та розмір векторних інструкцій в останніх поколіннях пристроїв задля прискорення обробки важких у обчислені проблем, таких як алгоритми машинного навчання. Це векторні інструкції, що належать до сімейства AVX-512 [23], з підвищеною точністю змін у слові та підтримкою операцій з плаваючою крапкою. Окрім основних виробників, є також більш спеціалізовані, такі як Epiphany [24]. Цей процесор спеціального призначення розроблений з архітектурою MIMD, яка використовує масив процесорів, кожен з яких має доступ до тієї самої пам'яті, прискорює виконання операцій з плаваючою крапкою. Це швидше, ніж надання кожному процесору власної пам'яті, оскільки спілкування між процесорами дороге. Найновішою мікросхемою виробника Adapteva є Epiphany V, що містить 1024 ядер на одній мікросхемі [25]. Хоча Adapteva ще не опублікувала специфікації споживання електроенергії Epiphany V, вона опублікувала результаті, які вказують на використання енергії всього 2 Вт [26].

Межі між традиційними суперкомп'ютерами та хмарою все більше розмиваються, коли мова йде про найкращі умови для виконання таких складних навантажень, як машинне навчання. Наприклад, GPU та прискорювачі зараз частіше зустрічаються в основних хмарних центрах обробки даних [27]. Як результат, паралелізація завантаженості машинного навчання стала першорядною для досягнення прийнятних показників у великих масштабах. Однак при переході від централізованого рішення до

розподіленої системи застосовуються типові завдання розподілених обчислень у вигляді продуктивності, масштабованості, стійкості до відмов або безпеки [28].

1.1.2 Горизонтальне масштабування

Хоча існує багато різних стратегій для збільшення потужності обробки однієї машини для широкомасштабного машинного навчання, є причини віддати перевагу горизонтальному масштабному проектуванню (рисунок 1.2) або поєднати два підходи, як це часто спостерігається в HPC. Перша причина - це менша вартість обладнання, як з точки зору початкових інвестицій, так і з обслуговування. Друга причина - стійкість до відмов, тому що, коли один процесор виходить з ладу в додатку HPC, система все ще може продовжувати роботу, ініціюючи часткове відновлення (наприклад, на основі контрольних точок, керованої комунікації [29], або часткового повторного обчислення [1]). Третя причина - збільшення сукупної пропускної здатності вводу/виводу порівняно з одним вузлом [30]. Навчання моделей - це завдання, що займає великі обсяги даних, і прийом даних може стати серйозним вузьким місцем [31]. Оскільки кожен вузол має виділену підсистему вводу/виводу, масштабування - це ефективна методика зменшення впливу вводу/виводу на продуктивність навантаження шляхом ефективного паралельного зчитування та запису серед декількох вузлів. Основна проблема масштабування полягає в тому, що не всі алгоритми машинного навчання піддаються розподіленню обчислювальної моделі, яка може бути використана лише для алгоритмів, які можуть досягти високого ступеня паралелізму.

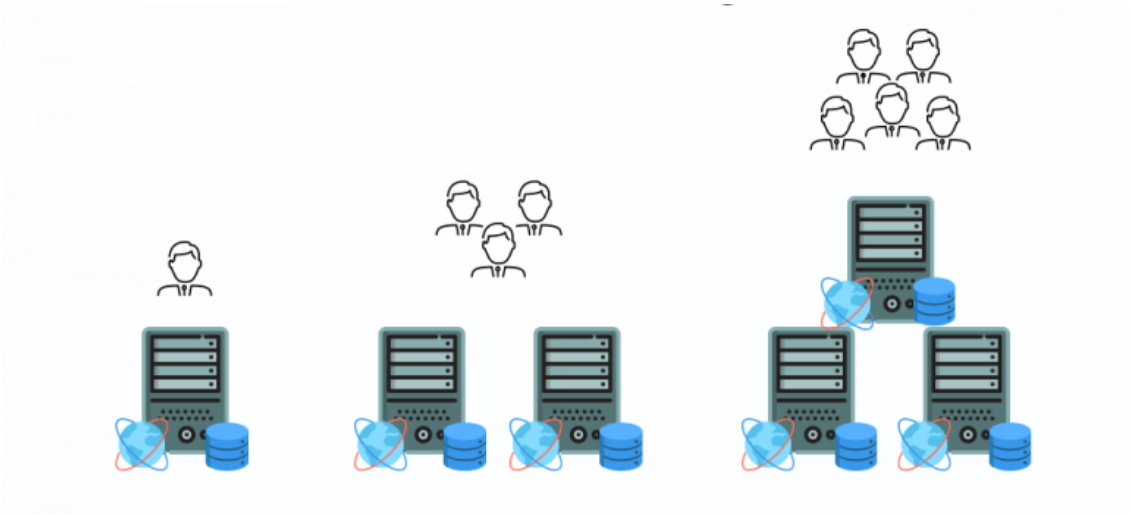

Рисунок 1.2 – Графічне представлення горизонтального масштабування

1.2 Підходи до розподіленої обробки даних

Розподілені системи для обробки великої кількості даних значною мірою покладаються на використання декількох малих серверів (рисунок 1.3), кожен з яких має відносно невелику ємність зберігання та обчислювальну потужність, а не один дорогий великий сервер. Ця стратегія виявилася більш доступною в порівнянні з використанням більш дорогого спеціалізованого обладнання. Одними з перших, хто впровадив такий підхід, є компанія Google, яка разом з цим ввела «коефіцієнт відмовостійкості» [32]. Впроваджений підхід дозволяє масштабувати систему таким чином, що сукупна пропускна здатність вводу/виводу є більшою у порівнянні з використанням меншої кількості великих потужних серверів. Пояснюється це тим, що кожен вузол має власну підсистему вводу-виводу, що у свою чергу дозволяє приймати більше вхідних даних, а отже оброблювати більше навантаження на систему у цілому [33].

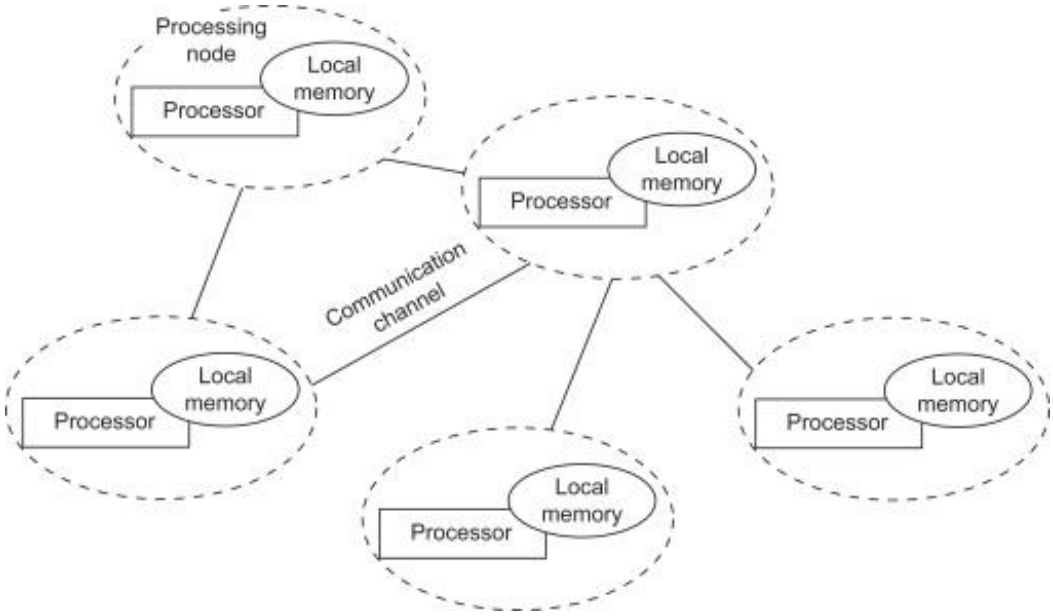

Рисунок 1.3 – Схема використання декількох робочих вузлів у розподіленій системі

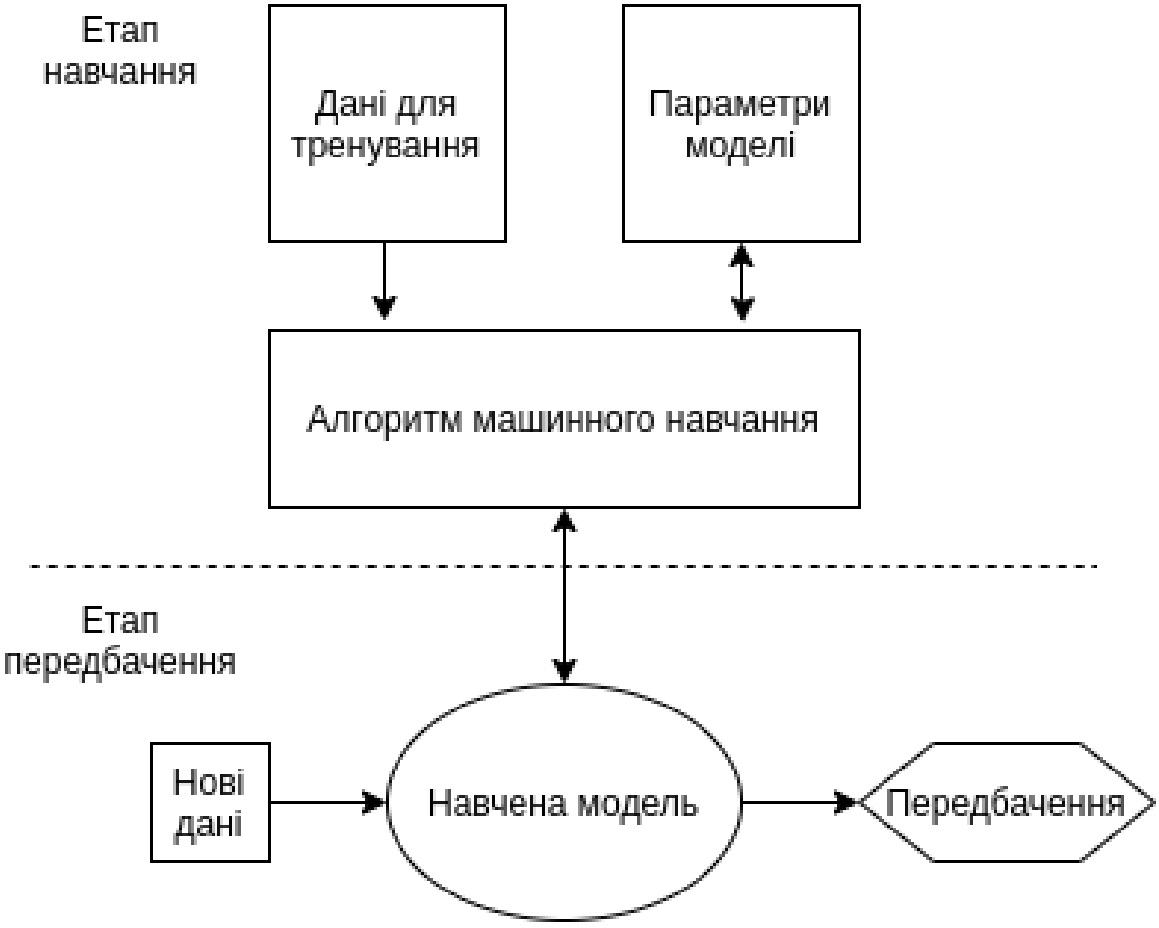

Рисунок 1.4 – Етапи машинного навчання

Розробка загальної системи, яка дозволяє ефективно розподілити машинне навчання, є складним, оскільки кожен алгоритм має чітку схему зв'язку [9] [34] [35]. Загалом кожну задачу машинного навчання можна розділити на етап навчання та етап прогнозування (рисунок 1.4).

Навчальний етап передбачає підготовку моделі машинного навчання шляхом подачі до неї великого масиву даних для навчання та ітеративне оновлення моделі за допомогою алгоритму машинного навчання. Окрім вибору відповідного алгоритму для окремо взятої задачі, необхідно знайти оптимальний набір гіперпараметрів для обраного алгоритму. Остаточний результат навчального етапу - це навчена (натренована) модель, яку потім можна розгорнути та використовувати для формування прогнозів (передбачень майбутніх результатів). Навчена модель приймає нові дані як вхідні та формулює прогноз як вихід. Хоча навчальна етап, як правило, обчислювально інтенсивніший та складніший, і вимагає наявності великих наборів даних, етап прогнозування можна виконати з меншою обчислювальною потужністю.

Описані вище етапи не взаємовиключні. Так зване інкрементальне навчання або донавчання поєднує етапи навчання та етапи прогнозування та постійно навчає модель за допомогою нових даних із фази прогнозування.

1.2.1 Розбиття даних

Якщо говорити про розбиття однієї великої задачі машинного навчання на дрібніші і виконання цих задач на різних вузлах системи, то існує два принципово різних підходу до цього питання: паралелізм рівня даних (англ. *Data-Parallel*) та паралелізм рівня моделі (англ. *Model-Parallel*) [36]. Ці два способи можуть застосовуватися одночасно [37].

1.2.1.1 Паралелізм рівня даних

У підході з паралелізмом на рівні даних (рисунок 1.5) дані розподіляються між усіма робочими вузлами системи, і всі робочі вузли

згодом застосовують один і той же алгоритм до різних наборів даних. Однакова модель доступна для всіх робочих вузлів централізовано (у такому випадку кожен з вузлів має виконувати додаткові звернення мережею задля зчитування моделі), або через реплікацію (кожен робочий вузол зберігає локальну копію моделі задля зменшення мережевого навантаження), задля формування єдиного та узгодженого результату. Такий підхід можна застосовувати до кожного алгоритму машинного навчання з припущенням незалежного та однакового розподілу щодо наборів даних (тобто до більшості алгоритмів машинного навчання [37]).

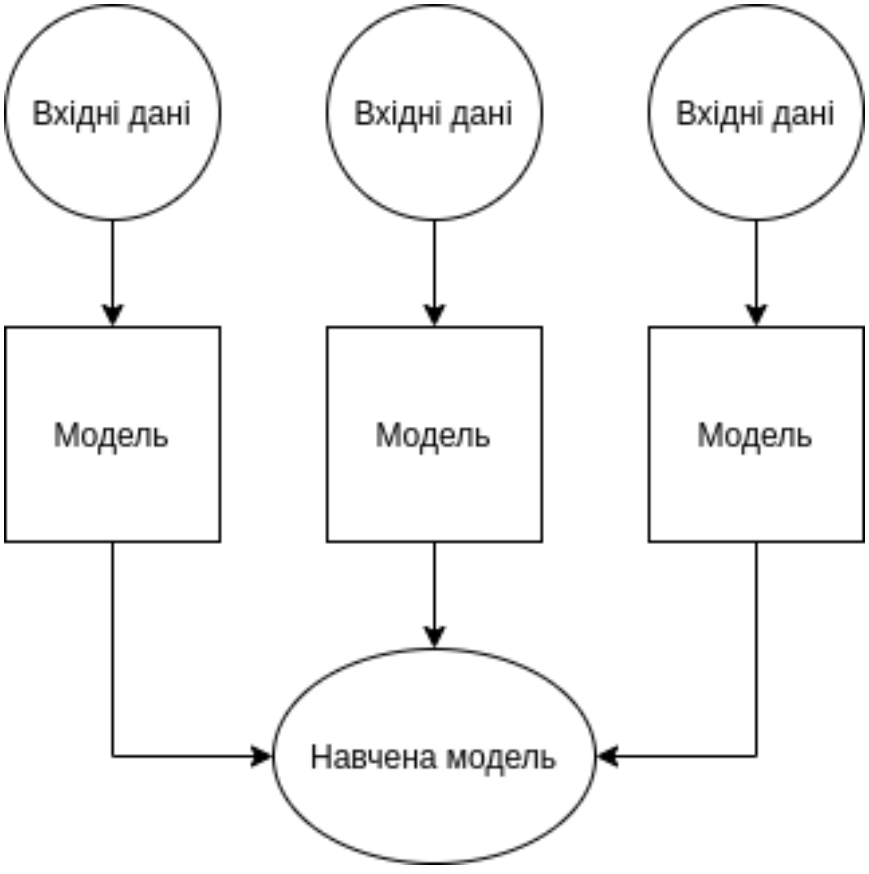

Рисунок 1.5 – Паралелізм рівня даних

1.2.1.2 Паралелізм рівня моделі

У підході з паралелізмом на рівні моделі (рисунок 1.6) точні копії всіх наборів даних обробляються робочими вузлами, які працюють на різних частинах моделі. Тому модель - це сукупність усіх моделей з кожного

робочого вузла. Паралелізм на рівні моделі не може автоматично застосовуватися до всіх алгоритмів машинного навчання, оскільки не завжди параметри моделі можуть бути розділені. Один зі способів вирішення цього - підготовка різних копій та версій однієї або подібної моделі та об'єднання результатів усіх підготовлених моделей, використовуючи методики, такі як збірка (англ. ensemble).

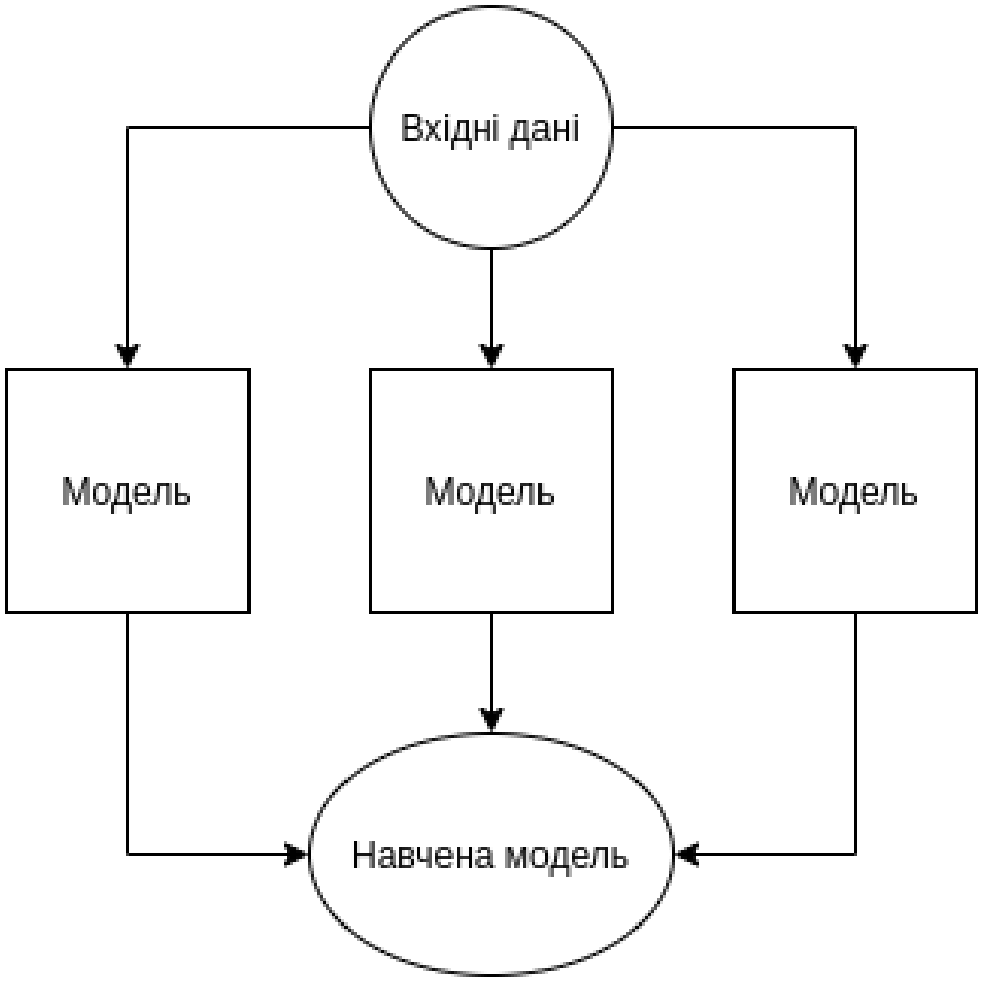

Рисунок 1.6 – Паралелізм рівня моделі

1.2.2 Збереження даних

Більша частина існуючих інструментів для розподіленого обчислення у питанні збереження даних орієнтуються на файлову систему Google (GFS) [38] або схожих реалізаціях. GFS належить Google і використовується для вирішення всіх потреб пов’язаних зі зберіганням великих об’ємів даних. GFS

розбиває дані, які завантажуються в кластер, на блоки, які потім розподіляються по вузлам системи (рисунок 1.7). Кожен з блоків має фіксований розмір, та має декілька копій серед інших вузлів системи, тобто реплікується. Такий підхід з копіюванням потрібен для гарантій відмовостійкості та доступності даних (коли один вузол вийде з ладу, то його дані будуть доступні на іншому, який може продовжити обчислення). Отримати доступ до даних можна шляхом звернення до головного вузла (англ. master node), який займається збереженням та зчитуванням блоків даних.

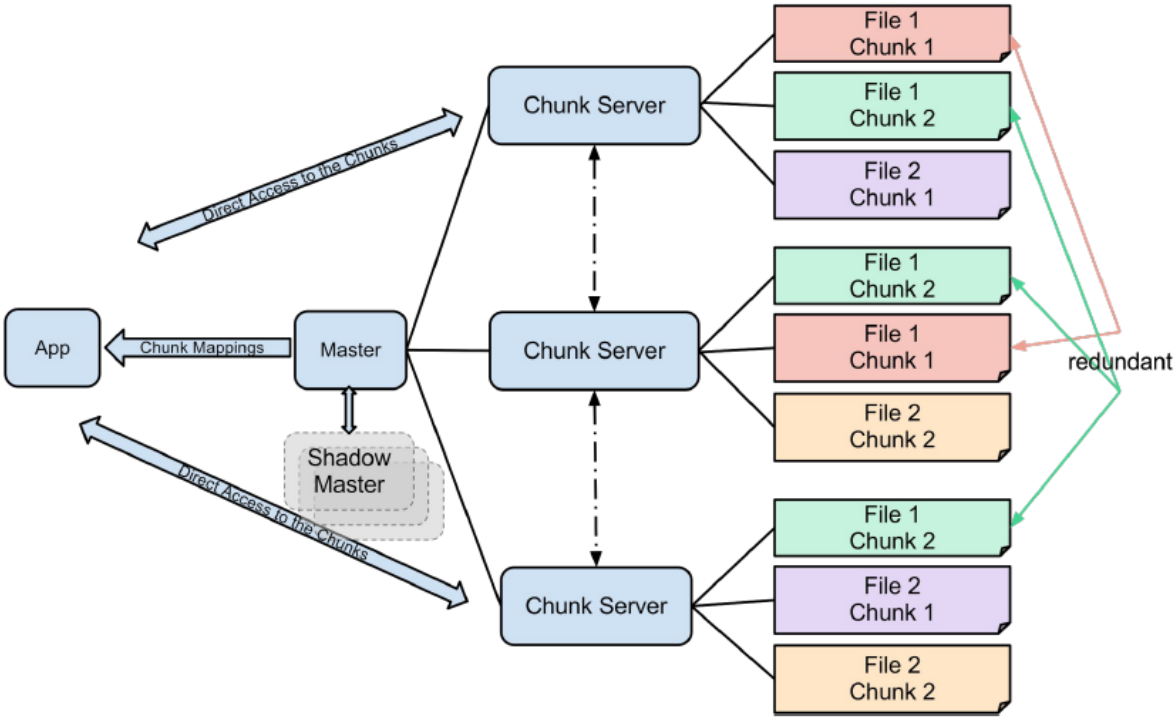

Рисунок 1.7 – Схема розбиття та збереження даних у GFS

Архітектура GFS була перейнята інструментом Hadoop [39]. У Hadoop використовує власну файлову систему Hadoop File System або HDFS [40], що є фактичною копією GFS з незначними відмінностями та покращеннями.

1.2.3 Проведення обчислень

Незважаючи на те, що архітектура пам’яті зводиться до блокової моделі, існує багато підходів до розподілення задач між вузлами та, власне, проведення обчислення.

1.2.4 MapReduce

Це інструмент і базова архітектура для обробки даних у розподілених системах, що була розроблена Google [41]. Архітектура складається з декількох фаз і запозичує концепції з функціонального програмування. Загальна схема зображена на рисунку 1.8.

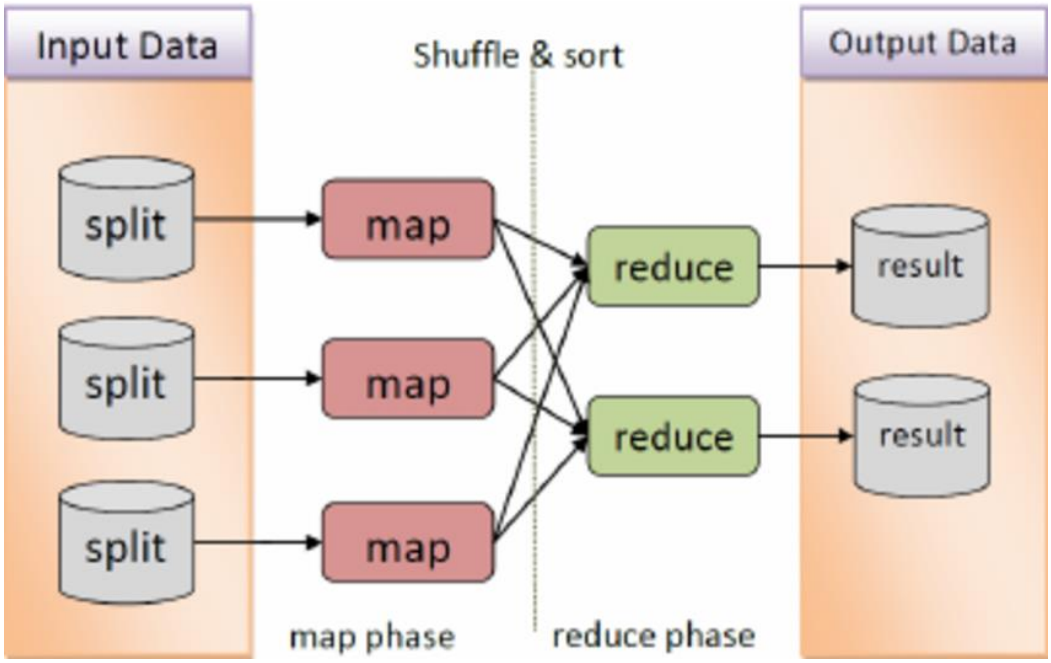

Рисунок 1.8 – Схема роботи MapReduce

У першій фазі, що називається map, всі дані розділяються на кортежі (пари ключ-значення). Це можна порівняти із відображенням функції другого порядку до набору у функціональному програмуванні. Фаза map може бути виконана повністю паралельно, оскільки між відображенням функції на два різні значення в наборі немає залежностей між даними.

Під час другої фази, що називається shuffle, ці кортежі обмінюються між вузлами. Це вкрай необхідно, оскільки агрегація результатів, як правило, залежить від даних на окремо взятому вузлі, і повинно бути гарантовано, що всі кортежі, належать до того ж ключа, обробляються тим же вузлом.

На останній фазі, фазі reduce, агрегацію проводять на кортежах для отримання єдиного вихідного значення по ключу. Однак, останній етап виконати паралельно неможливо, оскільки, оскільки кожен крок агрегації залежить від попереднього. Розбиття даних на малі блоки та їх перемішування є чинником паралелізму на перших двох етапах. У цьому і головна ідея підходу.

Основна перевага цього інструменту полягає в тому, що дані можуть розподілені на великій кількості обчислювальних вузлів і завдання кожної з фаз незалежні і тому можуть виконуватися цілком паралельно. Вузли системи можуть використовувати файлову систему GFS (або подібних), так що замість переміщення даних до коду програми код програми може бути переміщений до даних задля збільшення локальності даних та підвищення продуктивності. Код програми, у свою чергу, значно менше за розмірами для передачі мережею. Крім того, відповідно до ідеї масштабування, MapReduce реалізує відмовостійкість на рівні кластеру, контролюючи стан робочих вузлів, розподіл та перепланування завдань.

MapReduce як підхід був запропонований Google. Однак архітектура, що стоїть за підходом, відтворений у вищезгаданому Hadoop. Він використовує HDFS там, де MapReduce використовує GFS, але подібний за своєю загальною архітектурою.

1.2.5 Apache Spark

MapReduce та Hadoop сильно покладаються на розподілену файлову систему на кожному етапі виконання. Навіть проміжні результати зберігаються на рівні файлової системи, що призводить до величезного навантаження на фізичні накопичувальні пристрої, яким потрібно

неодноразово отримувати доступ до одних і тих же даних. Перетворення в лінійній алгебрі, як це відбувається у багатьох алгоритмах машинного навчання, мають типовий ітеративний характер. Крім того, парадигма операцій map та reduce не є ідеальною для підтримки послідовного потоку даних, що представляють собою ітеративні завдання [42].

Інструмент Apache Spark був розроблений як вирішення такої проблеми. Він здатний виконувати послідовність перетворень, як map, так і reduce, повністю в пам'яті [43]. Завдяки своїй структурі Spark може бути значно швидшим, ніж MapReduce, для високих навантажень. Якщо, наприклад, потрібні дві послідовні операції map, то потрібно буде виконати два окремих завдання MapReduce, кожна з яких має здійснювати читання з диску, запис проміжних та кінцевих результатів на диск. Spark у свою чергу зберігає всі дані в оперативній пам’яті, що заощаджує дорогі операції читання з диска.

Структуру даних, яку використовує Spark, називається Resilient Distributed Dataset (RDD). Такі набори даних доступні лише для читання, а нові екземпляри можна створювати лише із збережених даних на диску або шляхом трансформації існуючих RDD [44]. Кожному RDD присвоюється лінійний граф, який показує, які перетворення були виконані на ньому. Цей графік перетворень гарантує, що якщо деякі дані втрачені, Spark може простежити шлях, який пройшов екземпляр RDD та перерахувати втрачені дані. Важливо, щоб граф не містив циклів (тобто це спрямований ацилічний граф). Інакше Spark наткнеться на нескінченний цикл і не зможете відтворити RDD. На практиці необхідність повторного обчислення внаслідок втрати даних через збій вузла може призвести до пульсаційних ефектів[44]. Spark дозволяє проводити контрольну перевірку даних для запобігання повторного обчислення. Контрольні перевірки повинні бути викликані і матеріалізувати проміжний стан під час обрізання графа RDD. Такі системи, як TR-Spark [45], автоматизували генерацію контрольних точок, щоб Spark

міг працювати прозоро для користувача та всієї системи, коли переривання виконання повинно вважатися нормою.

Apache Spark також включає MLlib, масштабовану бібліотеку машинного навчання. Він також пропонує декілька утиліт для побудови процесів машинного навчання, впровадження часто використовуваних функцій перетворення, налаштування гіперпараметрів тощо.

1.2.6 Message Passing Interface

Message Passing Interface (MPI) [46] – інтерфейс взаємодії між вузлами системи за допомогою повідомлень, який призначений для високоефективних розподілених обчислень. MPI пропонує багато примітивів (наприклад, примітиви надсилання, отримання, транслювання та збору повідомлень), що дозволяють користувачам реалізовувати широкий спектр програм, включаючи алгоритми машинного навчання. Однак, завдяки своєму низькому рівню, реалізація багатьох завдань машинного навчання за допомогою MPI часто досить трудомістка і схильна до помилок, оскільки розробники повинні явно керувати такими аспектами, як розподіл даних та відмово стійкість.

1.2.7 Розподілений ансамбль методів

Багато універсальних інструментів та бібліотек машинного навчання мають обмежену підтримку розподіленості, хоча вони швидкі та ефективні на одному пристрої чи робочому вузлі. Один із способів досягти розподілу з використанням цих інструментів - це підготовка окремих моделей для окремих наборів даних з загального набору. Результати можуть бути об'єднані за допомогою стандартної агрегації у методі ансамблю [47].

Моделі, які дотримуються цієї стратегії, не залежать від конкретної бібліотеки. Їх можна оркеструвати, використовуючи існуючі інструменти розподілення (наприклад, MapReduce [41]). Навчальний процес включає паралельне навчання індивідуальних моделей на незалежних вузлах. Ні

оркестрація, ні спілкування не потрібні для початку навчання. Навчання на m вузлах з m підмножинами даних призводить до різних результуючих моделей. Кожна з них може використовувати окремі параметри або навіть алгоритми.

На час прогнозування всі підготовлені моделі можуть бути запущені з новими даними, після чого результат кожної з них агрегується. Це можна ще раз розподілити за потреби.

Важливим недоліком є те, що цей метод залежить від належного формування малих наборів даних для навчання. Якщо у навчальних наборах деяких моделей є великі упередження, ці екземпляри можуть спричинити необ'єктивний вихід ансамблю. Якщо дані розмічаються вручну, то перш за все необхідно забезпечити незалежність і однаковий розподіл даних. Існує велика кількість готових інструментів для цього методу, оскільки можна використовувати будь-які інструменти машинного навчання. У деяких популярних реалізаціях використовуються Tensorflow [4] та PyTorch [48].

1.2.8 Паралельний синхронний стохастичний градієнт

Синхронізований паралелізм часто є найпростішим для програми та міркувань. Існуючі бібліотеки розподілу (наприклад, інтерфейс передачі повідомлень MPI [6]) зазвичай можуть бути використані для цієї мети. Більшість підходів покладаються на операцію AllReduce, де обчислювальні вузли розташовані у деревоподібній топології. Спочатку кожен вузол обчислює значення локального градієнта, накопичує їх зі значеннями, отриманими від дочірніх вузлів і надсилає їх до батьківського. Врешті-решт, кореневий вузол отримує загальну суму і передає це вниз до дочірніх вузлів. Кожен вузол оновлює свою локальну модель щодо отриманого глобального градієнта.

1.2.9 Паралельні асинхронні стохастичні градієнти спуску

Асинхронні підходи, як правило, є складнішими для здійснення, і це може бути складніше простежити та налагодити поведінку під час виконання. Однак асинхронізм усуває багато проблем, які виникають у кластерах з високим рівнем відмов або непослідовною продуктивністю через відсутність частих бар'єрів синхронізації.

1.2.10 Хмарні обчислення

Багато хмарних провайдерів додають машинне навчання як одну зі своїх послуг. Більшість провайдерів пропонують кілька варіантів виконання завдань машинного навчання в їхніх хмарах, починаючи від послуг на рівні IaaS (віртуальні машини з попередньо встановленим програмним забезпеченням для машинного навчання) до рішень на рівні SaaS (машинне навчання як сервіс). Значна частина запропонованих технологій - це стандартні розподілені системи машинного навчання та бібліотеки.

Серед іншого, Google Cloud Engine Learning Engine пропонує підтримку TensorFlow і навіть надає екземпляри TPU [49]. Microsoft Azure Machine Learning дозволяє розгорнути модель через Azure Kubernetes, через пакетну службу або за допомогою CNTK VM [50]. Як конкурент TPU від Google, Azure підтримує прискорення використання програм ML через FPGA [51]. Amazon AWS представив SageMaker - сервіс для побудови та навчання моделей машинного навчання у хмарі. Послуга включає підтримку TensorFlow, MXNet та Spark [52]. IBM поєднала свої технології хмарного навчання на машині під брендом Watson [53]. Послуги включають Jupiter Notebooks, Tensorflow та Keras.

Хмарна модель поставки рішень стає все більш важливою, оскільки вона спрощує вхід та початок використання рішень та методів машинного навчання. Однак хмара є не лише споживачем розподілених технологій машинного навчання, але й підживлює розробку нових систем та підходів до

екосистеми для того, щоб вирішувати задачі масштабного розгортання більшої кількості вузлів.

1.3 Алгоритми та методи машинного навчання

Незважаючи на велике різноманіття алгоритмів та підходів до машинного навчання, використовувані представлення даних схожі між собою за структурою. Більшу частину обчислень складають операції з векторами, матрицями та тензорами, які у свою чергу є доволі відомими проблемами лінійної алгебри [4][5]. Перетворення в лінійній алгебрі зазвичай мають ітеративний характер, що дозволяє розглядати варіант з переходом від однопотокових алгоритмів до паралельних. Такий підхід часто може бути найскладнішим, оскільки він залежить від алгоритму і вимагає чіткого його розуміння.

Оскільки кількість і розмаїття алгоритмів і підходів настільки велика, що потребує окремого дослідження, то у цій роботі зосередимо увагу лише на тих, які можуть бути використані для вирішення задач розподілення у рамках магістерського диплому.

1.3.1 Еволюційні алгоритми

Такі алгоритми вчаться ітераційно на основі еволюції (еволюційний підхід) . Модель, яка фактично вирішує проблему, представлена сукупністю властивостей, званих її генотипом. Продуктивність моделі вимірюється за допомогою оцінки, обчисленої за допомогою функції пристосóваності. Після підрахунку показника *fitness* всіх генерованих моделей, наступна ітерація створює нові генотипи, засновані на мутації та схрещуванні моделей, які дають більш точні оцінки. Генетичні алгоритми можуть бути використані для створення інших алгоритмів, таких як нейронні мережі, мережі вірувань, дерева рішень та набори правил

1.3.2 Алгоритми засновані на стохастичному градієнтному спуску

Характеризуються тим, що мінімізують функцію втрат, визначену на основі результатів моделі, пристосовуючи параметри моделі в напрямку негативного градієнта (багатозмінної похідної функції). Спуск градієнта називається стохастичним, оскільки градієнт обчислюється з випадково вибіркової підмножини даних тренувань. Це дуже великий клас алгоритмів який налічує такі відомі, як от: метод опорних векторів, перцептрони.

Останні представляють собою бінарні класи, які позначають вхідні вектори як «активні» або «неактивні» та призначають вагу для всіх вхідних даних, а потім підсумовують результати цих ваг та їх входу. Результат цього порівнюється з певним порогом для маркування. Алгоритми на основані на перцептронах зазвичай використовують всю групу навчальних даних для знаходження рішення, оптимального для всього набору. Вони є бінарними, а тому в основному використовуються для двійкового класифікації.

Ще одним представником алгоритмів з класу є нейроні мережі. Вони основані на перцептронах, які складаються з декількох шарів: вхідного шару, одного або декількох прихованих шарів та вихідного шару. Кожен шар складається з вузлів, з'єднаних з попереднім і наступним шарами через ребра з пов'язаними вагами (зазвичай їх називають синапсами). На відміну від звичайних перцептронів, ці вузли зазвичай застосовують на виході функцію активації для введення нелінійностей у розрахунки.

1.3.3 Машинне навчання на основі правил

Алгоритми, що відносяться до цього метода, використовують набір правил, кожне з яких представляє невелику частину проблеми. Ці правила зазвичай виражають умову, а також значення, коли ця умова виконується. Через чітке співвідношення «якщо так тоді так» («if-else») правила зволяться до простих інтерпретації порівняно з більш абстрактними типами алгоритмів машинного навчання, таких як нейронні мережі. До таких можна віднести навчання асоціативних правил (англ. *Association Rule Learning*) та дерева

рішень (англ. *Decision trees*). Наприклад, дерева рішень формують гілки на основі заданих правил. Подорож по дереву передбачає застосування правил на кожному кроці до досягнення кисневих елементів (листків) дерева. Цей лист представляє рішення або класифікацію для заданого вводу.

1.3.4 Тематичне моделювання

Це статистичні моделі пошуку та відображення смислових структур у великих та неструктурованих наборах даних, найчастіше застосовуються на текстових даних. До групи алгоритмів відносять латентне розміщення Діріхле, латентний семантичний аналіз, наївний баєсівський класифіктор. Наприклад семантичний аналіз створює велику матрицю документів і тем у спробі класифікувати документи щоб знайти зв’язки між темами. Семантичний аналзі передбачає розподіл Гаусса за темами та документами. У свою чергу баєсівський класифікатор є імовірнісним класифікатором, що припускає незалежність різних ознак. Їх можна швидко навчити за допомогою контрольованого навчання, але вони менш точні, ніж складніші підходи.

1.3.5 Алгоритми матричної факторизації

Алгоритми можуть застосовуватися для виявлення прихованих факторів або виявлення відсутніх значень в структурі матричних даних. Наприклад, багато систем рекомендацій базуються на матричній факторизації матриці рейтингу користувальницьких елементів для пошуку нових елементів, які можуть зацікавити користувачів, враховуючи їх рейтинг щодо інших елементів [54]. Аналогічно факторизування лікарської сполуки цільової білкової матриці використовується для відкриття нових лікарських засобів [55]. Оскільки ця проблема співвідноситься з $O\ (F^3)$ з F розмірністю ознак, останні дослідження звертають увагу на масштабуванні цих методів на більші розміри характеристик[56].

1.3.6 Порівняння підходів та методів розподіленого машинного навчання

Різноманіття підходів та інструментів машинного навчання не завжди гарантує наявність усіх необхідних компонентів для виконання розподіленого навчання. У таблиці 1.1 наведені найголовніші критерії, які висуваються до системи, яка виконує розподілене навчання.

Існуючі рішення з використанням мови програмування Python є зручними у використанні, але не дозволяють досягати реального паралелізму через обмеження середовища, у якому запускається код навчання. Проте, такі інструменти як Numpy і Tensorflow дозволяють виконувати обрахунки на графічних процесорах, що дозволяє досягати бажаних результатів навіть не виконуючи розподілення.

Такі бібліотеки як MPI гарно виконують розподілення, але не заточені під виконання ітераційних лінійних перетворень. У свою чергу Apache Hadoop та Apache Spark вирішують такі задачі, оскільки обчислення виконуються у пам’яті робочих вузлів і результати записуються безпосередньо на жорсткі диски, що забезпечує відмовостійкість та можливість виконувати обчислення незалежно на кожному з вузлів без виконання додаткових обмінів повідомлень.

Логічним є використання сучасного рушія Apache Spark оскільки він ідеально підходить під виконання розподілених обчислень, підтримує потокову обробку та швидкий за рахунок використання оперативної пам’яті на відміну від Apache Hadoop де кожна дія записується на фізичний диск. Використання оперативної пам’яті не лише зменшує навантаження на пристрої читання/запису, а й дозволяють швидше проводити розрахунки лінійної алгебри.

Таблиця 1.1 – Порівняння підходів та методів машинного навчання

Особливість	Numpy	Apache Spark	Apache Hadoop	Tensorflow	MPI
Розподіленість	-	+	+	-	+
Паралелізм рівня даних	-	+	+	+	+
Паралелізм рівня моделі	-	+	+	+	-
Відмовостійкість	-	+	+	-	-
Підтримка SQL	+	+	+	-	-
Масштабованість	-	+	+	-	+
Підтримка обчислень на графічному процесорі	+	+	+	+	+
Потокова обробка	-	+	-	-	-

1.3.7 Постановка завдань дослідження

Задачу наукової роботи можна сформулювати як дослідження методів розподіленого навчання на прикладі одного з алгоритмів машинного навчання. Пропонується розглянути алгоритм Isolation Forest (дерево ізоляцій). Для досягнення поставленої мети необхідно вирішити такі задачі: сформувати набір даних для навчання, виконати навчання у середовищі з одним обчислювальним вузлом та у розподіленому середовищі з декількома обчислювальними вузлами, виконати порівняльне дослідження, дослідити ефективність підходу розподілення.

Висновки та постановка завдань дослідження

Розподілене машинне навчання - процвітаюча екосистема з різноманітними рішеннями, які відрізняються архітектурою, алгоритмами, продуктивністю та ефективністю. Доводиться подолати деякі основні проблеми, щоб зробити розподілене машинне навчання життєздатним в першу чергу, наприклад, знайти механізми для ефективної паралелізації

обробки даних при одночасному поєднанні результатів в єдину цілісну модель. Існує ще багато відкритих викликів, які мають вирішальне значення для довгострокового успіху розподіленого машинного навчання. До таких можна віднести повноцінну підтримку розподіленого навчання, збільшення та підтримку відмовостійкості системи, пошук методів до розподілення без зменшення ефективності навчання та зменшенні мережевої взаємодії.

Задачу наукової роботи можна сформулювати як дослідження методів розподіленого навчання на прикладі одного з алгоритмів машинного навчання. Пропонується розглянути алгоритм Isolation Forest (дерево ізоляцій). Для досягнення поставленої мети необхідно вирішити такі задачі: сформувати набір даних для навчання, виконати навчання у середовищі з одним обчислювальним вузлом та у розподіленому середовищі з декількома обчислювальними вузлами, виконати порівняльне дослідження, дослідити ефективність підходу розподілення.

2 ПІДХІД ДО РОЗПОДІЛЕНОГО МАШИННОГО НАВЧАННЯ НА ПРИКЛАДІ ПОШУКУ АНОМАЛІЙ

Машинне навчання має на меті навчання на наборах даних, що відображають певну предметну область. Узгодив формат та набори даних для алгоритму машинного навчання потрібно вирішити наступні задачі: розбити етап навчання на підзадачі, які можуть бути виконанні незалежно одне від одного, розподілити ці задачі між наявними робочими вузлами у розподіленому середовищі, виконати необхідні обчислення та перетворення, зібрати результати з робочих вузлів та звести всі результати до однієї моделі. Для формалізації задач використаємо математичний апарат дискретної математики, лінійної алгебри, машинного навчання на основі правил та розподіленого обчислення.

2.1 Задача пошуку аномалій

Під час аналізу реальних наборів даних існує загальна потреба визначати які об'єкти з набору значно різняться за інші. Такі об’єкти і називаються аномаліями.. Пошук аномалій - це завдання знаходження та ідентифікації таких аномальних об’єктів з загального набору даних [57], що має важливе застосування у широких областях, наприклад, для виявлення мережевих атак у кібербезпеці, для виявлення шахрайських транзакцій у сфері фінансів та виявлення захворювань у сфері охорони здоров’я. На рисунку 2.1 зображений графік довільною функції, що монотонно спадає та зростає протягом часу. Точці O_1 властивий перепад у зоні локального мінімуму, що не є характерним для інших проміжків. На рисунку 2.2 наведені приклади пошуку аномалій різними алгоритмами. Як бачимо, точки синього кольору є аномальними і не належать до загальної або спільних груп (кластерів) як точки червоного кольору.

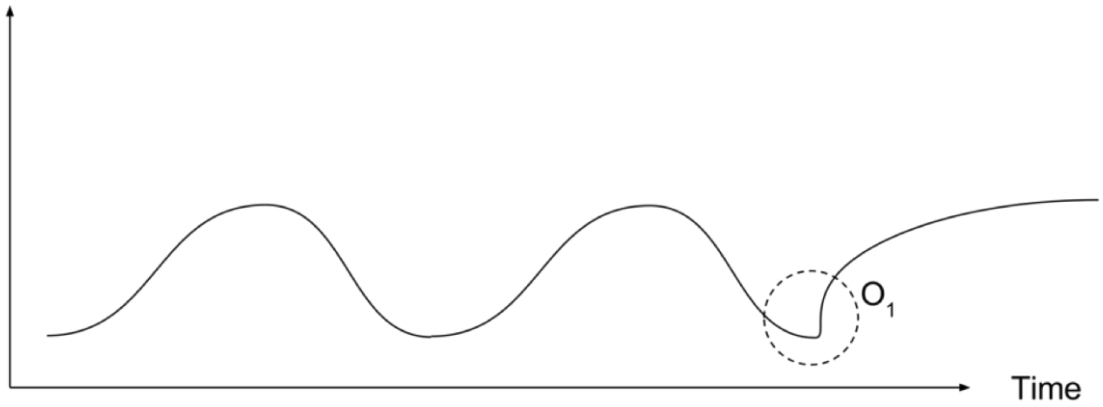

Рисунок 2.1 – Приклад аномалії для довільною функції, що монотонно спадає та зростає протягом часу у зоні локального мінімуму

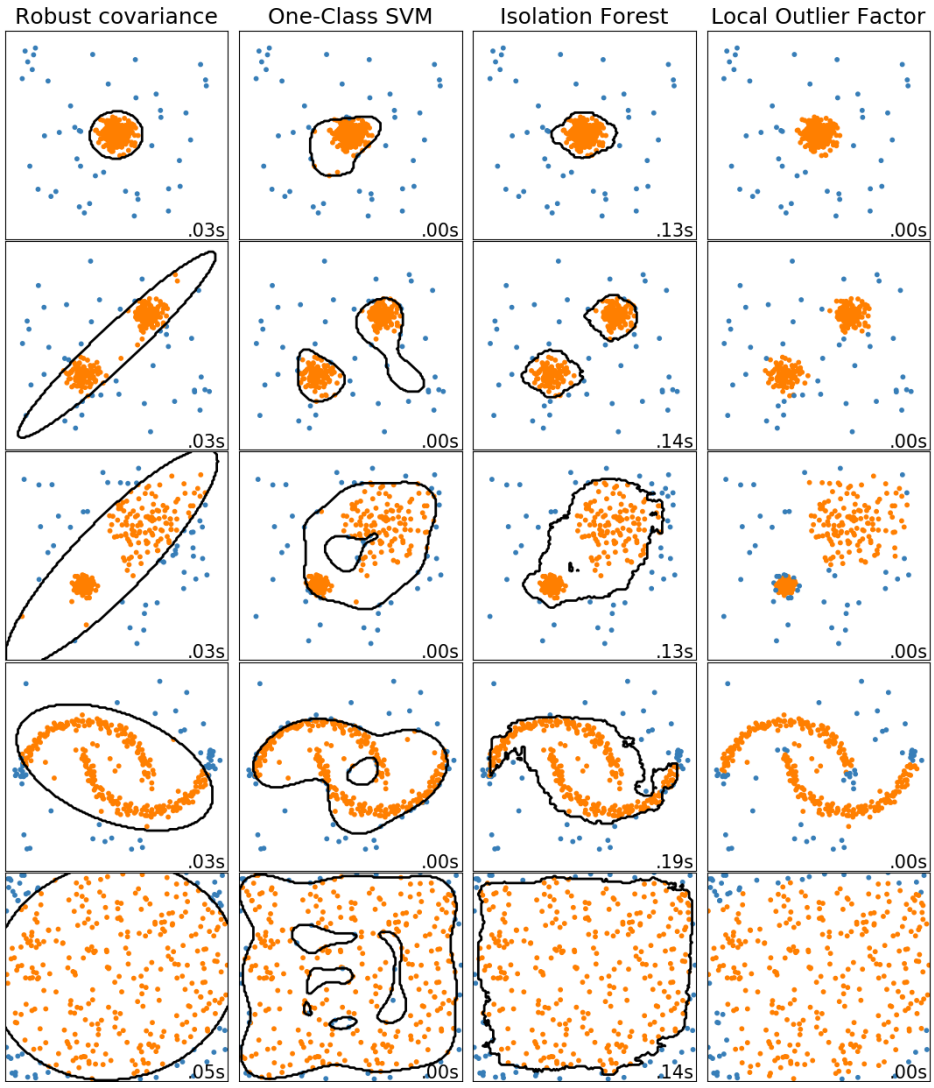

Рисунок 2.2 – Визначення аномалій різними алгоритмами машинного навчання. Точки синього кольору є аномаліями

У загальному виді, задачу пошуку аномалій можна сформулювати наступним чином: нехай заданий вектор чи матриця ознак $X \in R^{N*D}$, де N – кількість елементів у вхідному наборі даних, а D – кількість ознак для кожного за таких елементів. Наприклад, N може бути кількістю зображень у деякій колекції зображень і D – кількість пікселів, що використовуються для представлення зображення [58]. Тоді задача пошуку аномалій зводиться до визначення елемент чи рядок з X є аномальним, тобто значно відрізняється з поміж інших елементів чи рядків.

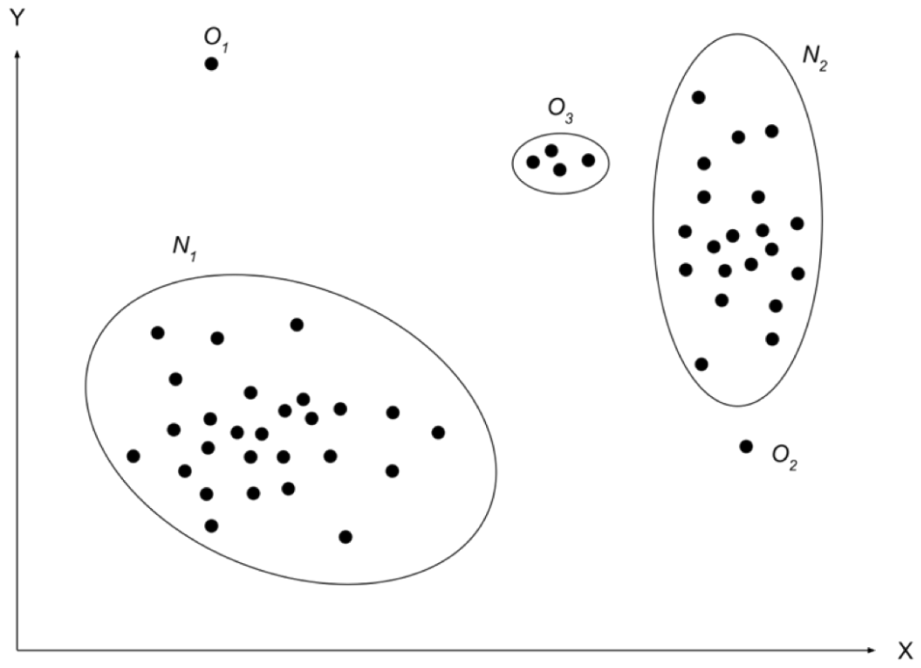

Рисунок 2.3 – Пошук аномалій у двовимірному просторі. Набори точок N_1 та N_2 можна об'єднати у дві окремі групи, O_1 та O_2 є аномаліями, набір точок O_3 на можна однозначно віднести до аномалій чи окремої групи, потрібно більше даних

Якщо розглядати пошук аномалій у тексті, то вектор ознак може містити такі ознаки як: емоційне забарвлення тексту, належність до певної категорії тощо, відсоток вмісту певних мовленнєвих конструкцій тощо.

2.2 Дерево ізоляцій

В даній роботі досліджується вирішення задачі розподіленого навчання на прикладі алгоритму дерев ізоляцій [59]. Для початку визначимося з тим що таке дерево ізоляцій.

Дерево ізоляцій, яке буде описане нижче, має багато чого спільного з фундаментальною структурою даних як двійкове дерево пошуку. Дуже спрощено, двійкове дерево пошуку - це особливий вид деревних структур, де ключі зберігаються в такому порядку, що пошук вузла здійснюється шляхом ітераційного (або рекурсивного) вибору лівої чи правої гілки на основі кількісного порівняння (наприклад, меншої чи більшої)). Вставка нового вузла (рисунок 2.4) виконується шляхом пошуку у дереві, використовуючи описаний раніше метод, до досягнення зовнішнього вузла, куди буде вставлений новий вузол. Це дозволяє здійснювати ефективні пошуки вузлів, оскільки в середньому половина дерева не буде відвідуватися. Для ілюстрації цього припустимо значення $x = [1,10,2,4,3,5,26,9,7,54]$ є відповідними вставками у дерево. Потім проміжні кроки будуть такими, як показано нижче. Проміжні кроки зображені на рисунку.

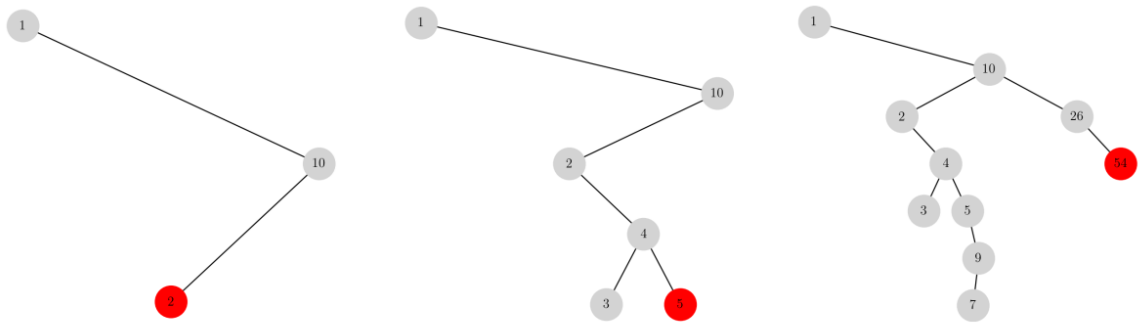

Рисунок 2.4 – Вставка нового вузла у двійкове дерево пошуку

Однією з властивостей дерева двійкового пошуку є те, що з випадковим чином генерованими даними шлях між кореневим вузлом та зовнішніми вузлами, як правило, буде коротшим. На рисунку 2.5 нижче ми

бачимо, що шлях від 7 вдвічі перевищує довжину, ніж для підозрілого значення 54. Ця властивість буде грати важливу роль в алгоритмі ізоляції.

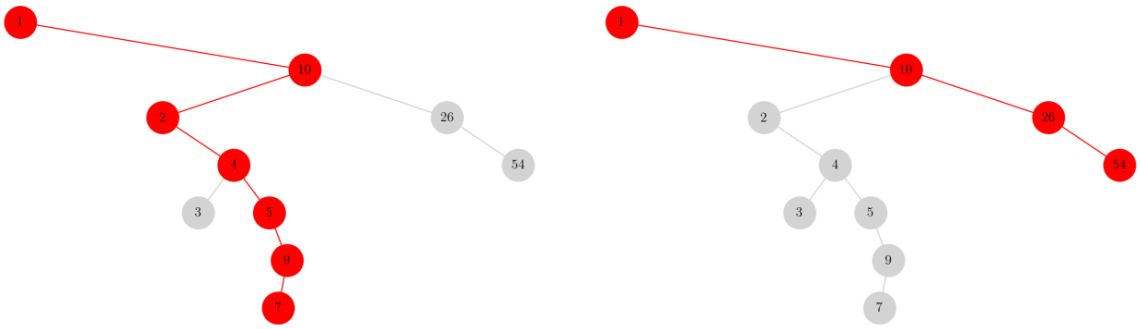

Рисунок 2.5 – Приклад пошуку у двійковому дереві пошуку

Нехай T – вузол у дереві ізоляцій. T може бути як зовнішнім вузлом без дочірніх вузлів, так і внутрішнім з одним тестом (англ. test) та тільки двома дочірніми вузлами (T_l, T_r). Тест складається з атрибуту q та роздільника p таким чином що тест $q < p$ ділить елементи даних на T_l та T_r.

Нехай заданий набір даних $X = \{x_1, \dots, x_n\}$ з n елементів з багатовимірного нормального розподілу. Для побудови дерева ізоляцій ми рекурсивно розбиваємо множину X на випадково обрані атрибути q та роздільник p до тих пір поки не виконаються одна з умов:

– Дерево досягло межі своєї висоти.
– $|X| = 1$.
– Всі дані з набору X мають однакові значення.

Дерево ізоляцій - це власне бінарне дерево, де кожен вузол у дереві має рівно два дочірні вузли або не має жодного. Припускаючи, що всі елементи з набору даних унікальні, то кожен елемент ізольований від зовнішнього вузла таким чином, що кількість зовнішніх вузлів дорівнює n, а кількість внутрішніх – $n - 1$; загальна кількість вузлів дерева - $2n - 1$; і, отже, потреба в пам'яті обмежена і зростає лінійно з n.

Довжиною шляху будемо називати $h(x)$ елементу x, що вимірюється кількістю ребер які проходить x деревом з кореневого вузла поки прохід не буде завершеним у зовнішньому вузлі.

2.3 Пошук аномалій методом лісу ізоляцій

Показник аномалій потрібен для будь-якого методу визначення аномалій [59]. Складність у виведенні такої оцінки з $h(x)$ полягає у тому, що хоча максимально можлива висота дерева ізоляцій зростає з порядком n, середня висота дерева зростає з порядком $\log(n)$. Нормалізація $h(x)$ будь-яким з перерахованих вище засобів або не обмежена, або не може бути безпосередньо порівняна. Оскільки дерева ізоляцій мають структуру, еквівалентну двійковому дереву пошуку (як було описано вище), оцінка середнього $h(x)$ для зовнішніх вузлів така ж, як і оцінка невдалого пошук у двійковому дереві пошуку. Таким чином використовуємо аналіз двійкового дерева пошуку для оцінки середньої довжини шляху у дереві ізоляцій. Вирахуємо середню довжину шляху невдалого пошуку у двійковому дереві для набору даних з n елементів:

$$c(n) = 2H(n-1) - (2(n-1)/n) \quad (1)$$

де $H(i)$ - гармонічне число, яке можна оцінити як $\ln(i) + 0{,}5772156649$ (константа Ейлера). Оскільки $c(n)$ є середнім значенням $h(x)$ для заданого n, ми використовуємо його для нормалізації $h(x)$. Оцінка аномалії s елементу x визначається як:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \quad (2)$$

де $E(h(x))$ – середнє значення $c(n)$ з набору ізольованих дерев. З рівняння (2) слідує, що:

- коли $E(h(x)) \to c(n), s \to 0.5$;
- коли $E(h(x)) \to 0, s \to 1$;
- і коли $E(h(x)) \to n-1, s \to 0$ s монотонна до $h(x)$.

Використовуючи оцінку аномалії, ми можемо зробити наступні висновки:

– якщо елементи повертають s дуже близькими до 1, то вони є безумовно, аномаліями;

– якщо елементи мають s значно менше 0,5, то їх цілком безпечно вважати звичайними елементами;

– якщо всі екземпляри повертають $s \approx 0{,}5$, то вся вибірка не може мати жодної чіткої аномалії.

Контур з показників аномалії може бути отриманий шляхом передачі тестового набору через колекцію дерев ізоляції (ліс ізоляцій), що сприяє детальному аналізу результатів виявлення. На рисунку 2.6 наведений приклад такого контуру, що дозволяє користувачеві візуалізувати та ідентифікувати аномалії в просторі елементів. Використовуючи контур, ми можемо чітко визначити три точки, тут $s \geq 0{,}6$, які є потенційними аномаліями.

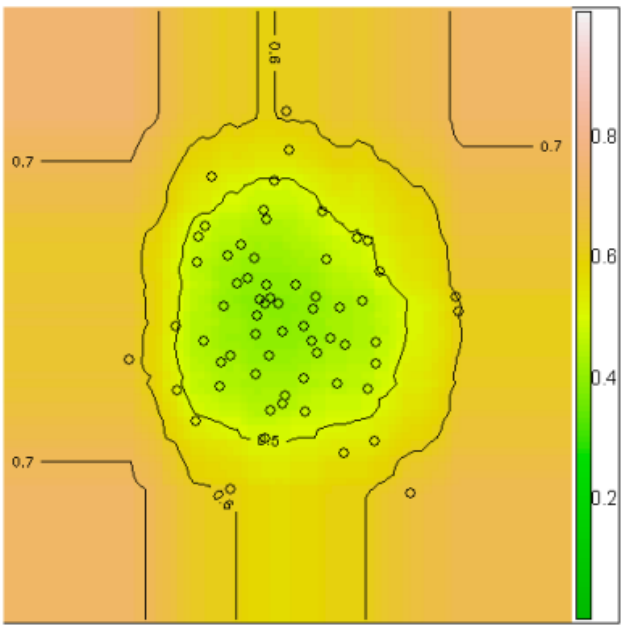

Рисунок 2.6 – Контур показників аномалій на лісі дерев ізоляцій для розподілу Гаусса з 64 елементів. Проілюстровані лінії контуру для $s = 0{,}5, 0{,}6, 0{,}7$. Потенційні аномалії можна виділити як точки, де $s \geq 0{,}6$

2.4 Підхід до розподіленого навчання

Виявлення аномалії за допомогою дерев ізоляцій є двоступеневим процесом. Перший (навчальний) етап будує дерева ізоляції за допомогою піднаборів з загального навчального набору. Другий етап (тестування) проганяє тестові елементи з наборів через дерева ізоляції, щоб отримати оцінку аномалії для кожного елементу.

Оскільки кожне дерево може навчатися незалежно на випадкових наборах даних, то логічним є спроба виконати розподілення таких обчислень, враховуючи що кількість дерев та даних може бути великою, не зважаючи на те, що ріст складності оцінюється в n.

Вартим уваги э структура, у яку організовуються робочі вузли у кластері. Вирішальним фактором для топології є ступінь розподілу, яку система покликана реалізувати. Централізовані системи використовують суворо ієрархічний підхід до агрегації, що відбувається в одному центральному місці. Децентралізовані системи дозволяють проводити проміжну агрегацію або з реплікуваною моделлю, яка послідовно оновлюється, коли результати транслюється на всі вузли.

У конкретному випадку (рисунок 2.7) кожне дерево навчається незалежно і може бути обраховане на окремому вузлі, потрібно лише надати вузлу потрібні дані. У такому разі підходить централізована модель топології розподіленого навчання, де існує один вузол, який займається розбиттям на підзадачі та збіркою результатів до купи.

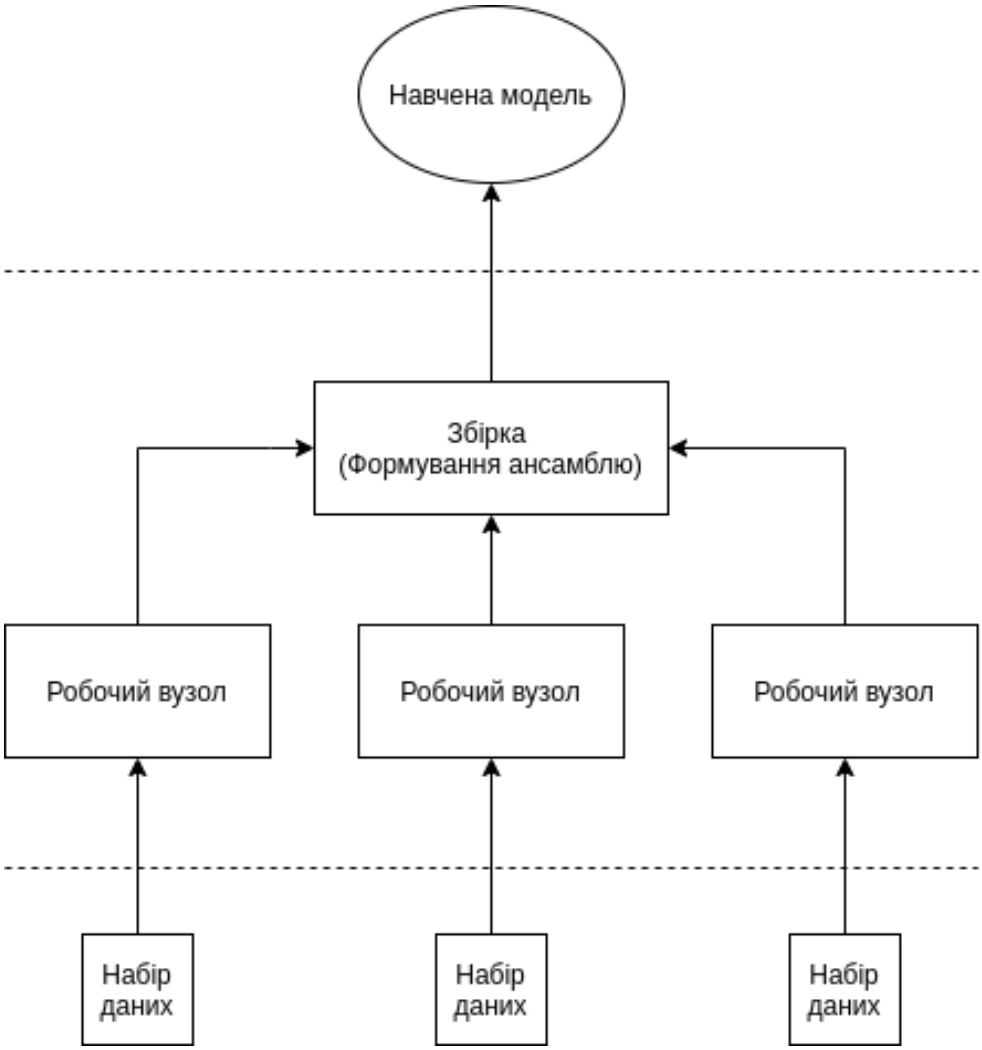

Рисунок 2.7 – Централізована модель розподіленого машинного навчання

Висновки до розділу

У даному розділі розглянуто та описано алгоритми та підходи, що будуть використовуватись для виконання розподіленого машинного навчання на прикладі алгоритму дерев ізоляцій. Розглянуті підходи застосовуються для вирішення задачі пошуку аномалій. Як видно з опису, робота з деревами ізоляцій зводиться до роботи з двійковими деревами пошуку та може бути розподілена між декількома робочими вузлами системи.

3 РОЗРОБКА МЕТОДУ РОЗПОДІЛЕНОГО МАШИННОГО НАВЧАННЯ

Для вирішення задачі розподіленого машинного навчання на великих наборах даних та у розподіленому середовищі пропонується використовувати паралелізм на рівні даних [41], оптимізувати виконання окремих задач на вузлах системи, залучити потокову обробку [58], виконувати вертикальний розподіл.

3.1 Паралелізм рівня даних

Алгоритм з використанням дерев ізоляцій для побудови дерев використовує не весь набір даних, а деякі його частини, тобто деякі вектори ознак. Варто відзначити, що з обраних векторів ознак можуть використовуватися не всі ознаки для побудови конкретного екземпляра дерева, що змушує нас задуматися о доцільності використання всіх ознак у навчанні окремого дерева, та передачі таких зайвих і невикористовуваних даних мережею.

У навчальному процесі обчислювальні завдання коефіцієнта посилення кожної змінної функції займають більшу частину навчального часу. Однак для цих завдань потрібні лише дані поточної змінної функції та цільової змінної функції. Тому для зменшення обсягу даних та вартості передачі даних у розподіленому середовищі пропонується вертикальний підхід до розподілу даних для дерев ізоляцій відповідно до незалежності змінних функцій та ресурсних вимог обчислювальних завдань. Навчальний набір даних таким чином буде розділений на кілька підмножин. Таким чином ми зменшимо об’єми наборів даних, які приймають участь у навчанні окремих дерев.

Беручи за основу початковий набір даних, який складається з довільної кількості векторів ознак (рисунок 3.1) ми можемо сформувати менші набори

вибираючи лише ті стовпці $(c_1, \dots, c_N)$, які необхідні на поточному етапі навчання (рисунок 3. 2).

c1	c2	...	...	...	...	...	...	...	...	...	...	cN

Рисунок 3.1 – Початковий набір даних як список з векторів ознак

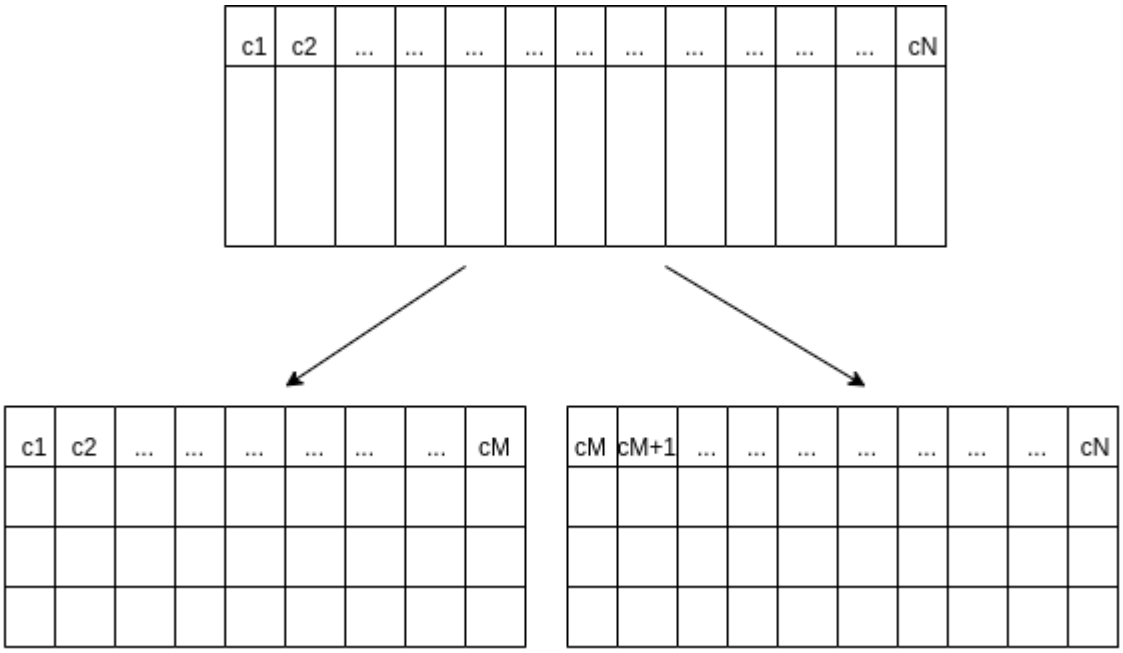

Рисунок 3.2 – Розбиття початкового набору даних на менші набори

Як результат, ми можемо сформувати більш менші набори даних шляхом копіювання не стовпців з векторів ознак, а лише одного стовпчика, що відповідає за приналежність конкретного вектору ознак до тієї чи іншої категорії з навчання. Враховуючи особливості використання технології Apache Spark, ми можемо графічно представити це у вигляді різних RDD

(Resilient Distributed Datasets)[44], що властиво лише для Spark (рисунок 3.3). Елементи y відповідають ознакам, а елементи x – самим векторам ознак.

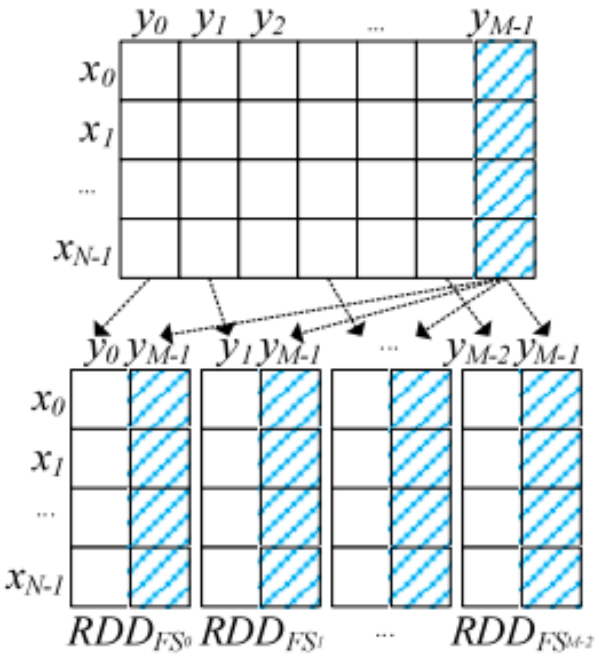

Рисунок 3.3 – Розбиття набору даних на менші і формування з них RDD об’єктів

Ще однією оптимізацією є підготовка та копіювання окремих наборів даних на робочих вузлах системи (рисунок 3.4), де кожен з наборів або його частина знаходяться як локальні копії разом з кодом, що виконує їх обробку. У такому випадку, кожен робочий вузол може швидше реагувати на запити головного вузла, і не витрачати час не передачу даних мережею. Цей підхід не є ефективним на малих наборах даних, оскільки об’єми можуть не досягати таких, які вимагають бути оптимізованими.

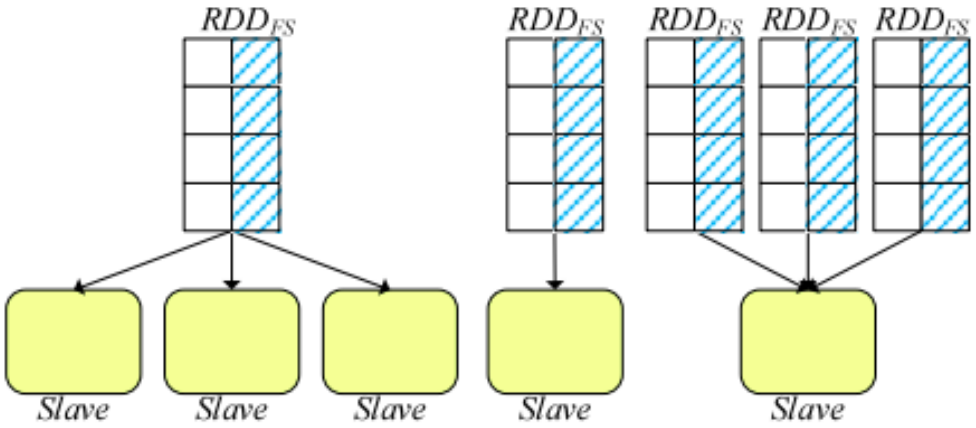

Рисунок 3.4 – Розподіл RDD об’єктів між робочими вузлами системи

3.2 Вертикальний розподіл

Відповідно до алгоритму дерев ізоляцій, кожне дерево є незалежним, групи таких дерев формують ліс дерев (ансамбль), серед яких вже узгоджується рішення щодо вхідних даних, наприклад чи вважати їх аномальними чи ні.

Формування спільного і єдиного дерева відбувається на головному вузлі, який саме і займається групуванням результатів з робочих. У свою чергу, робочі вузли займаються тим, що будують окремо взяті дерева. Процес групування займає багато часу у головного вузла і може призводити до простоїв робочих вузлів за рахунок того, що вузли очікують нових задач та нових наборів даних, що формуються зазвичай на головному вузлі централізовано.

Як вирішення такої проблеми, пропонується використовувати вертикальний розподіл та оптимізувати такий процес на рівні задачі де кожен з робочих вузлів не лише будує окремі взяті дерева, а й формує окремі ліси з таких дерев. З поданого на вхід набору даних формуються різні комбінації для окремо взятих дерев. Навчання кожного лісу можливо виконувати паралельно на робочому вузлі, якщо таке дозволяють обчислювальні ресурси. Сформований ліс відправляється на головний вузол, де будується загальний ліс.

Такий підхід дозволяє головному вузлу приймати рішення не лише на основі сформованих дерев, але й на основі сформованих ансамблів на різних наборах даних та використовує обчислювальні ресурси на повну робочу силу зменшуючи час простою. Загальна схема розподілу наведена на рисунку 3.5.

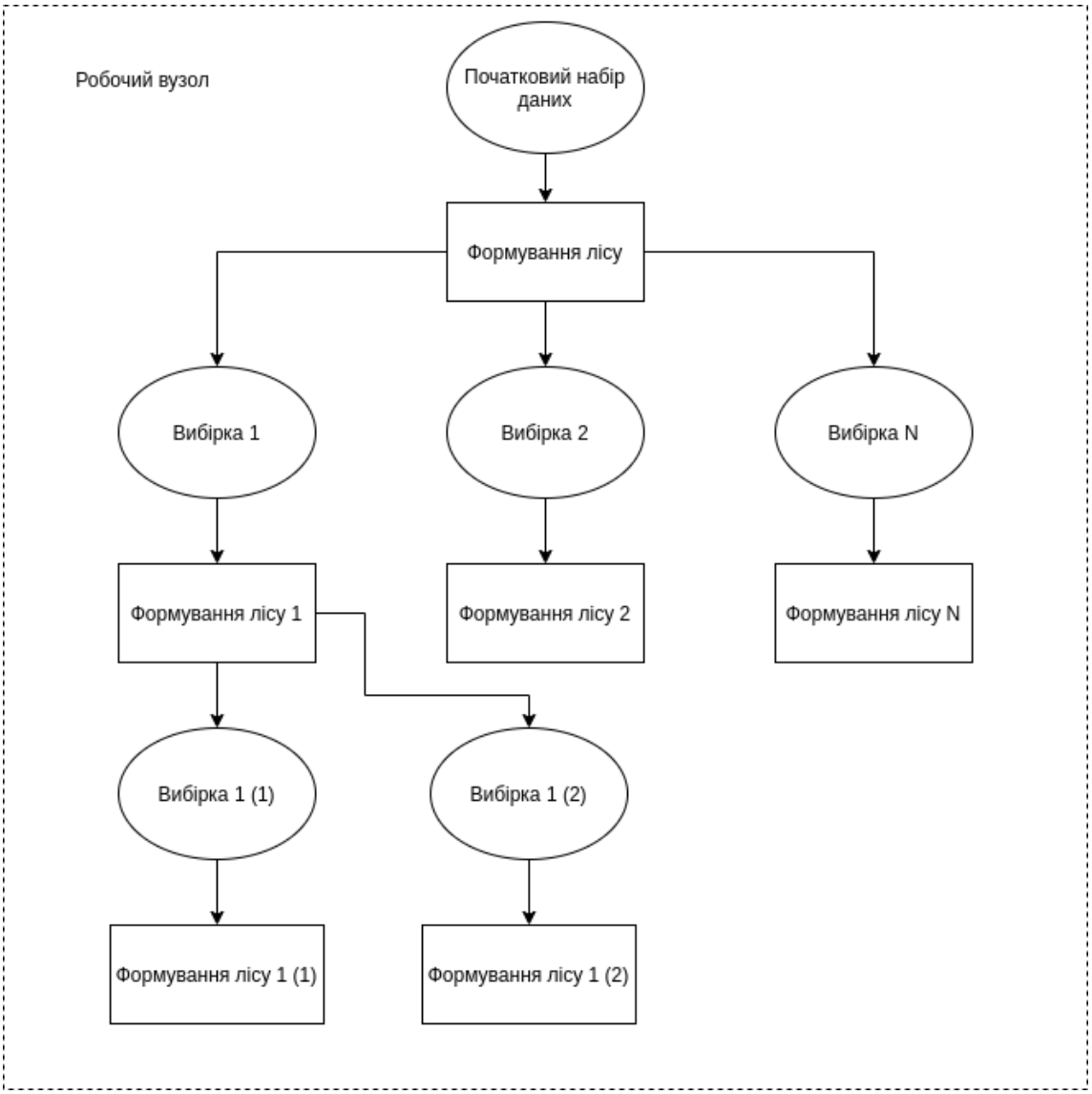

Рисунок 3.5 – Формування лісу дерев на робочих вузлах системи

3.3 Підготовка даних

Дані для навчання представляють собою набір векторів ознак та результат приналежності до аномалій того чи іншого елементу з набору. Результат приналежності є обов’язковим, оскільки на його основі можна перевірити точність навченої моделі. Оскільки, у роботі використовується алгоритм машинного навчання без учителя, то до подальших наборів даних

така перевірка може не застосовувати, що значно зменшить точність та правильність прогнозування навченої моделі.

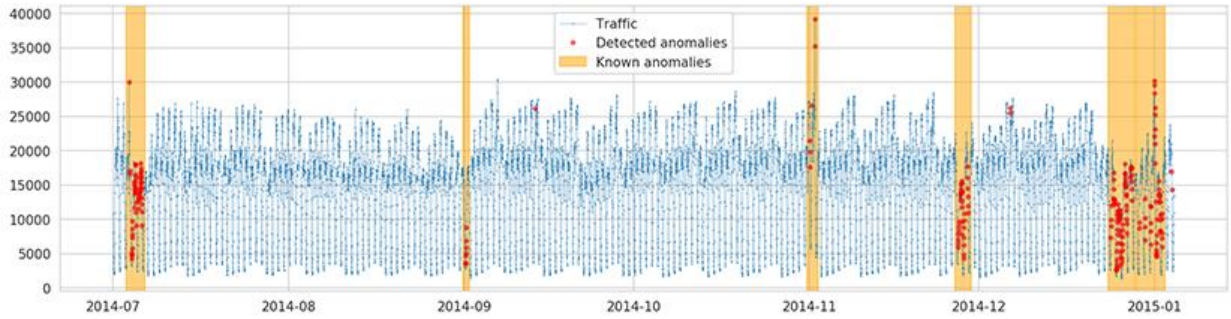

Рисунок 3.6 – Пошук аномалій у мережевій активності протягом часу

Початковий набір може взятий з реальних джерел шляхом збереження послідовності певних подій. Одним з прикладів є мережева активність протягом періоду часу (рисунок 3.6).

Так як пошук аномалій є задачею виявлення таких елементів з набору даних, що не підлягають загальним правилам та характеристикам, то основний набір можна згенерувати штучно, шляхом завищення або заниження певних ознак так, аби елемент з набору виділявся серед усереднених значень.

Основний же набор даних, на яких відбувається навчання та тестування, взятий з оригінальної статті методу пошуку аномалій на основі дерев ізоляцій. Ці набори даних вибрані, оскільки вони містять відомі класи аномалії як основну істину, і ці набори даних використовуються в літературах для оцінки детекторів аномалій в потокових даних [59].

До таких наборів належать: Http (KDDCUP99), Smtp (KDDCUP99), Shuttle.

3.4 Потокова обробка

Потокова обробка представляє собою опрацювання нескінченного набору даних у реальному часі. Це є вигідним коли потрібно оброблювати дані на льоту і видавати результати користувачу якомога швидше. Якщо

розглядати потокову обробку для машинного навчання, то нові набори даних, що надходять з потоку, можна використовувати для донавчання існуючої моделі, навчання окремої нової моделі, перевірки правильності результатів попереднього навчання тощо. У загальному вигляді потокову обробку можна представити як:

$$Z = \{z(1), z(2), \dots z(t), z(t+1), \dots\}$$

де $z(t) \in R^N$ для $t \geq 1$ і N – розмірність вектору ознак.

З використанням потокових даних виникає проблема навчання моделі на основі історичних даних та використання моделі для прогнозування нових даних, на які ми не знаємо відповіді. З технічної точки зору, таке прогнозування чи передбачення – це проблема наближення функції відображення f заданих вхідних даних X для прогнозування вихідного значення y, тобто визначення $y = f(X)$

У деяких випадках відносини між вхідними та вихідними даними можуть змінюватися з часом, тобто в свою чергу відбуваються зміни невідомої основної функції відображення. Зміни можуть бути наслідковими, наприклад, прогнози, зроблені моделлю підготовленою за старими історичними даними, вже не є правильними або такими ж правильними, як це могло б бути, якби модель навчалася на більш нових наборах даних. Прикладом можуть бути зміни у вподобаннях користувача на мережевих ресурсах. Зміни у його вподобаннях відображаються на значеннях вектору ознак, що призводить до зовсім інших вхідних даних для навчання моделі. Ми ж можемо відслідковувати такі зміни у наборах і за потреби виконувати донавчання моделі або повне перенавчання.

Багато методів вилучення даних припускають, що виявлені закономірності є статичними. Однак на практиці моделі даних розвиваються з часом. Це ставить перед собою дві важливі проблеми. Перша – виявлення дрейфу концепції (англ. *concept drift*) [60]. Друга – постійно оновлювати модель але без повного перенавчання моделі з самого початку.

3.4.1 Концепція Дрейфу

Поняття дрейфу [60] в машинному навчанні та обробці даних стосується зміни взаємозв'язків між вхідними та вихідними даними протягом часу. В інших областях цю зміну, можливо, називають «коваріатним зрушенням», «зрушенням набору даних» або «нестаціонарністю».

У більшості складних додатків, дані змінюються з часом і повинні аналізуватися майже в реальному часі. Зв’язки та відносини між такими даними еволюціонують, тому моделі, побудовані для аналізу таких наборів даних, швидко застарівають. У машинному навчанні та обробці даних це явище називають поняттям дрейфу. Як приклад можна навести зміну температури повітря у залежності від сезону. Або дії користувача у залежності від поточного економічного положення країни тощо. Крім того, наведені приклади вказують на те, як сильно модель залежить від предметної області де вона застосовується.

Зміна даних може мати будь-яку форму. Концептуально простіше розглянути випадок, коли існує деяка часова послідовність змін, що дані, зібрані протягом певного періоду часу, показують однаковий взаємозв'язок і що цей зв'язок з часом плавно змінюється. Проте, це не завжди так, і це припущення не є вірним, оскільки може включати поступову зміну, повторні або циклічні, раптові або різкі зміни. Для кожної ситуації можуть знадобитися власні методи виявлення дрейфу. Зазвичай, повторювані зміни та довгострокові тенденції вважаються систематичними і можуть бути чітко визначені та оброблені.

3.4.2 Способи виявлення та вирішення проблеми дрейфу

Зважаючи на те, що тенденція до дрейфу відбувається після розгортання моделі і використання її у реальних додатках, найкращий спосіб дії - це стежити за змінами та вживати заходів, коли зміни відбудуться.

Наявність циклу зворотного зв’язку від системи моніторингу та оновлення моделей з часом допоможе уникнути застарілості моделей.

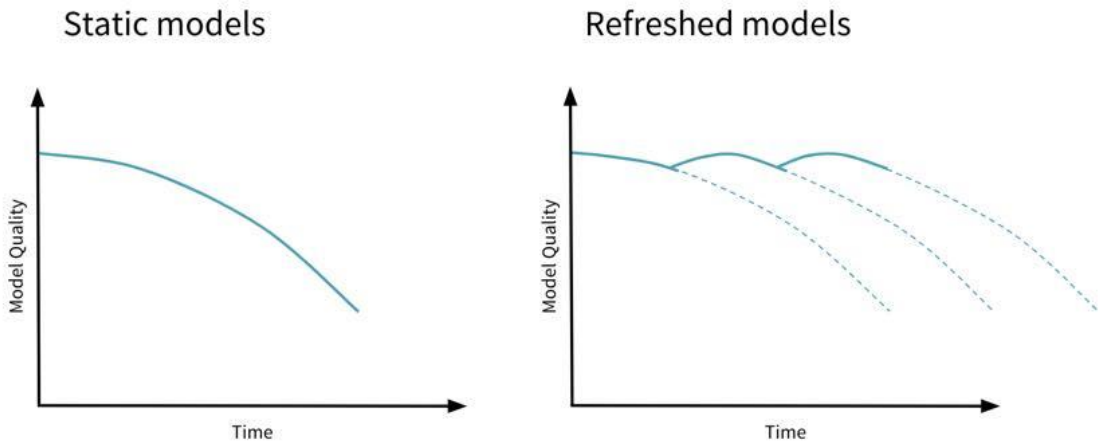

Рисунок 3.7 – Деградація навченої моделі протягом часу залежно від підходу до навчання

Як ми бачимо вище (рисунок 3.7), навчена модель схильна деградувати з часом. Вчасне виявлення таких спадів і реагування на них дозволить тримати модель у актуальному стані. Загалом існує декілька варіантів вирішення проблеми дрейфу:

1. Не робити нічого. Взагалі не обробляти та припускати, що дані не змінюються. Це дозволяє один раз розробити єдину "найкращу" модель і використовувати її на всіх майбутніх даних. Це має стати вашою відправною точкою та базою для порівняння з іншими методами.
2. Періодично донавчати модель актуальними даними.
3. Періодично оновлювати модель новою. Нова модель у свою чергу – це результат накопичення та навчання на основі нових даних, що вже не є статичними, як у початкової моделі.
4. Використовувати ансамблевий підхід. Може використовуватися там, коли статична модель залишається недоторканою, але нова модель вчиться виправляти прогнози статичної моделі на основі взаємозв'язків в

останніх даних, тобто наступні моделі корегують прогнози попередніх моделей.

3.4.3 Запропонований підхід до проблеми дрейфу

На початку система у своїй основі має заздалегідь навчену модель. Навчання відбувається на статичних даних, які перевіряються з очікуваними результатами. Такими очікуваними результатами є наявність аномалій або їх відсутність, тобто логічні «так» чи «ні». Якщо розглядати варіант класифікації, то результатами можуть бути певні класи, їх назви або ідентифікатори.

Маючи навчену модель та очікувані результати, ми можемо підрахувати відсоток неправильно визначених екземплярів u. Логічним є допустити, що чим менше цей показник, тим краще. Проте, з часом дані можуть змінюватися, що призводить до збільшення цього показника. У цьому разі, такі збільшення є сигналом нашій потоковій системі до перенавчання моделі. Оскільки перенавчання є довготривалим процесом і ми використовуємо потоки, то будемо виконувати навчання лише на нових наборах даних. Така нова навчена модель стає поточною моделлю, з якою будуть відбуватися наступні порівняння. Інакше, показник u є задовільним і створення нової моделі не є необхідним. Визначення проміжку та розміру набору даних, необхідних для створення нової моделі є складною задачею і має визначатися емпірично залежно від даних та предметної галузі до якої вони застосовуються.

Варто відзначити, що алгоритм, який використовується у дослідженні, а саме дерева ізоляцій, є алгоритмом машинного навчання без учителя, тобто для виконання навчання не потрібні заздалегідь розмічені дані на категорії і не потрібно значи чи цей екземпляр з набору є аномальним чи ні. Використання розмічених даних допомагає перевіряти модель на точність. Інакше, ми можемо навчити модель таким чином, що різні набори даних

будуть правильними як і моделі побудовані на їх основі, проте результати прогнозування будуть протилежні. Розглянемо це на прикладі дослідження аномалій у мережевому трафіку.

Припустимо, що здійснюється аналіз мережевого трафіку на наявність перепадів у навантажені на мережу. Такі навантаження можуть бути спричиненими DoS атаками. Очікуваною реакцією системи на такі атаки буде усунення джерела атаки та, по можливості, призупинення обробки таких мережевих запитів. Якщо говорити про потокову обробку, то початкова модель буде мати у своїх наборах більше доброякісних елементів, ніж злоякісних. Таким чином визначення аномалій зведеться до пошуку закономірностей у загальній вибірці, що призведе до правильного результату у визначенні аномалій. Якщо почнеться атака на систему і потоковий набір даних буде містити більше злоякісних елементів, то визначення аномальних екземплярів зведеться до пошуку доброякісних, що цілком не те, чого ми очікуємо. Описана ситуація вказує на те, що навчання у потокових системах може виконуватися правильно, проте крок перевірки результатів навчання може бути відсутнім. У цьому випадку наявність розмічених даних і використання їх у потокових наборах даних є вирішенням.

З іншого боку, ми можемо проігнорувати такі перевірки правильності, коли наша модель намагається аналізувати вподобання користувачів. Кожен користувач може починати роботу з системою з початковим набором вподобань, що відображені у статично навченій моделі. З часом, а коли ми говоримо про користувачів системи, то це може зводитися до декількох годин, вподобання можуть змінюватися, що призведе до застарілих і вже не актуальних рекомендацій. У цьому випадку, ми можемо перенавчати нашу модель не спираючись на перевірки її точності, оскільки самі користувачі визначають наскільки точною є модель. Такий підхід може звести модель до визначення зовсім протилежного результату, що буде коректним у розрізі поточної предметної області. Це рідкий випадок коли деградація системи призводить до покращення результатів.

Незважаючи на два таких протилежних приклади, ми можемо звести вирішення задачі потокового навчання до схеми зображеної на рисунку 3.8.

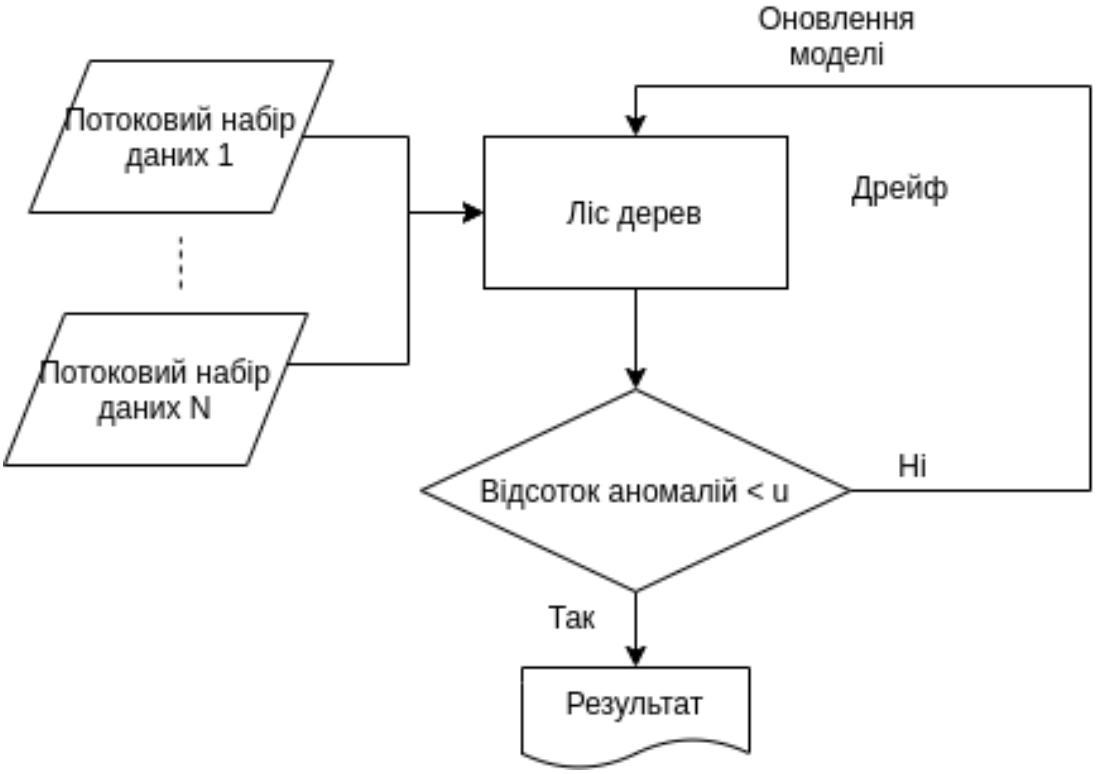

Рисунок 3.8 – Схема машинного навчання з використанням потокової обробки даних

Кожен набір даних потраплятиме на вхід до поточної моделі, якщо відсоток аномалій не перевищує допустимий, то залишаємо поточну модель, інакше виконуємо перенавчання на новому наборі і зберігаємо отриману навчену модель як поточну.

3.5 Розподілене навчання

Навчання відбувається на підготованому наборі даних, що завантажений на головний вузол системи. Обраним методом для навчання є дерева та ліс ізоляцій. Кожен з вузлів системи буде отримувати набір даних необхідний для навчання конкретного дерева. Побудоване дерево додається до загального набору дерев та використовуватиметься у визначенні аномалій. Сам процес побудови дерева не залежить від іншого такого процесу, тобто

може виконуватися паралельно і незалежно. Обробка потоків виконується схожим чином, за винятком того, що відбувається накопичування нового набору даних і подальше навчання на його основі.

У загальному вигляді алгоритм розподіленого навчання з використанням ізоляційних дерев можна розписати наступним чином:

Крок 1. Зчитати набір даних для навчання.

Крок 2. Сформувати матрицю, що складається з вектору ознак та категорій до яких належить кожен з векторів.

Крок 3. Для кожного такого набору ознак сформувати RDD об’єкт.

Крок 4. Запустити паралельну обробку RDD об’єкта методами Apache Spark.

Крок 5. Сформувати ізоляційне дерево та додати його до локального результату.

Крок 6. Повернути результат у вигляді набору дерев.

Крок 7. Відсортувати зібрані дерева задля формування лісу ізоляцій.

Крок 8. Сформувати ліс ізоляцій.

Крок 9. Додати його до загального результату.

Крок 10. Повторювати пункт 1 до завершення навчання на усіх наборах вхідних даних.

Крок 11. Здійснити перевірку точності навчання порівнявши отримання значення з очікуваними з початкових наборів.

Висновки до розділу

У розділі було розроблено підхід до розподіленого машинного навчання з використанням потоків даних на прикладі дерев ізоляцій. Початковий набір даних є статичним де кожний елемент представляє собою вектор ознак та має чітку приналежність до аномалії або відсутність такої приналежності.

Обробка відбувається потоково з поступовою деградацією навченої моделі протягом часу. Таке явище називається дрейфом. Для усунення такої проблеми пропонується виконувати донавчання моделі новими наборами якщо відсоток аномалій збільшується. Для оновлення актуальності моделі пропонується проводити перенавчання. Період накопичення нових даних та об’єми вибірок маються визначатися експериментально.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗПОДІЛЕНОГО МАШИННОГО НАВЧАННЯ

4.1 Огляд аналогів

Якщо розглядати життєвий цикл машинного навчання на високому рівні, то він складається з двох кроків:

– навчання моделей: на цьому кроці ми подаємо набори даних до алгоритму на вхід. Результат - аналітична модель;

– формування прогнозів: На цьому кроці ми використовуємо аналітичну модель для прогнозування нових результатів та подій на основі вивченого шаблону.

Машинне навчання - це безперервний процес, де ми з часом неодноразово вдосконалюємо та оновлюємо аналітичну модель [20]. Прогнозування може здійснюватися різними способами в рамках програми або сервісу. Один із способів - вбудувати аналітичну модель безпосередньо в додаток для обробки вхідних запитів як зображено на рисунку 4.1.

Рисунок 4.1 – Аналітична модель вбудована безпосередньо у застосунок

Крім того, розгортання аналітичної моделі може відбуватися на окремо виділеному сервері моделей та використовувати мережеві протоколи (HTTP або RPC) для зв’язку додатку з сервером (рисунок 4.2).

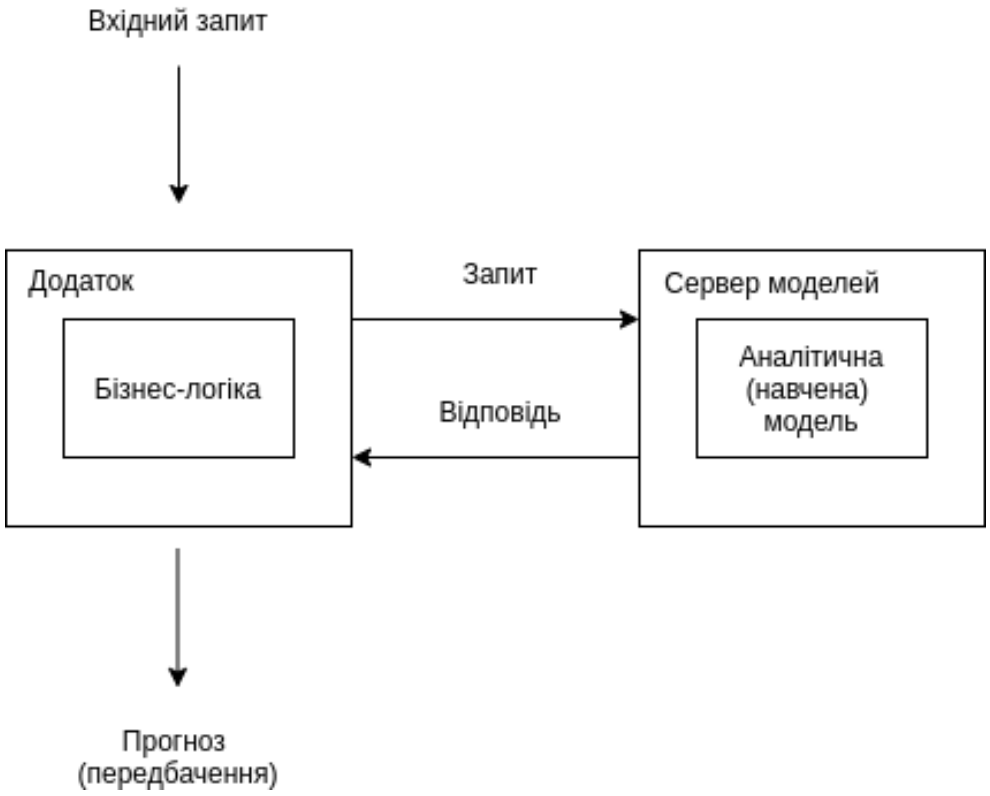

Рисунок 4.2 – Аналітична модель на окремому сервері моделей

Обидва варіанти [28] мають свої переваги та недоліки. До переваг використання окремого серверу моделей можна віднести:

– просту інтеграцію з існуючими технологіями та організаційними процесами;

– підтримка незалежного управління моделями, контроль версій;

– абстрагування від конкретних реалізацій того чи іншого сервісу.

Недоліками є:

– такий сервер часто прив’язаний до конкретних технологій машинного навчання або до постачальника хмарних рішень;

– більш висока затримка відповіді мережею;

– більш складні налаштування безпеки (віддалений зв’язок через брандмауер та управління авторизацією);

– відсутність автономної роботи у режимі оффлайн (без з’єднання та підключення до головного серверу моделей).

Схеми, зображені на рисунках 4.1 та 4.2, дають нам уявлення того, як навчені моделі будуть використовуватися безпосередньо сервісами, для яких такі моделі навчаються. У загальному вигляді, архітектура повноцінного процесу навчання виглядає так, як зображено на рисунку 4.3, і складається з джерел вхідних наборів даних, прикладних застосунків, модуля побудови модель машинного навчання та загального сховища даних, де зберігається все, що можна було зібрати з вхідних ресурсів та підготувати у якості вхідних параметрів до модуля машинного навчання [28].

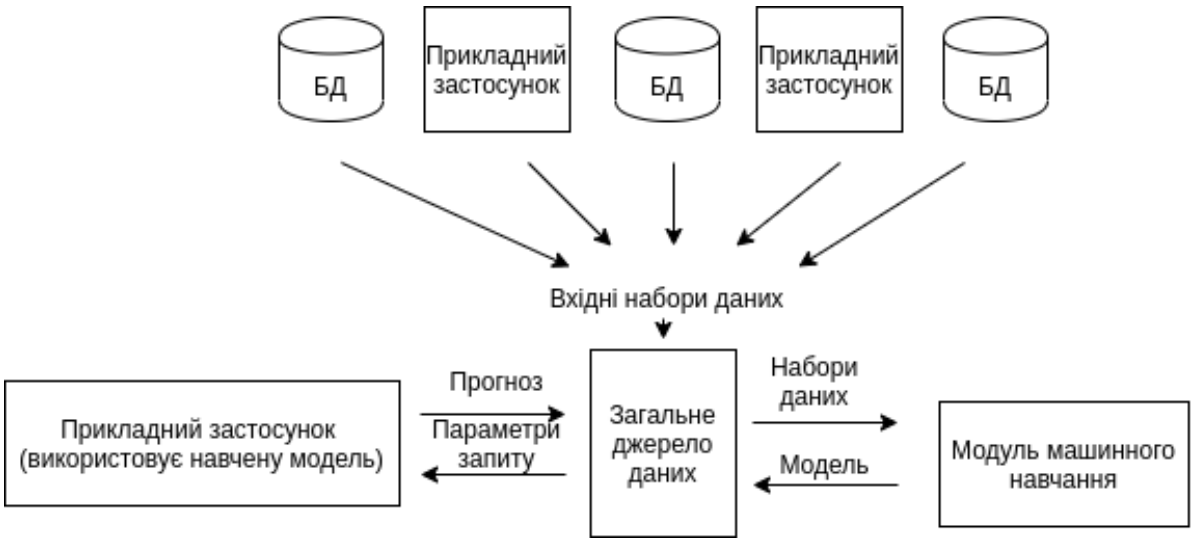

Рисунок 4.3 – Архітектура процесу машинного навчання

Описаний вище підхід застосований до більшості систем, що використовують машинне навчання у свої основі. Варто відзначити, що кожен модуль може бути розбитий на підмодулі і мати власні базові блоки та елементи для прискорення або покращення виконання навчання.

Дані можуть надходити з різних джерел. Це можуть бути як окремі бази даних, так і сервіси, які час від часу завантажують нові набори даних до

загального джерела, або навіть звичайні файли, які мають бути перенесені або скопійовані безпосередньо до джерела.

У якості загального джерела даних (англ. Data lake) виступає файлове сховище або спеціалізовані файлові системи як GFS. Дані зберігаються у довільному форматі та у довільних структурах, що вимагає подальшої обробки спеціалізованими інструментами, такими як Apache Hadoop [41] або Apache Spark [45]. У такому випадку, під обробкою розуміється трансформування сирих неструктурованих даних до певної структури, яка необхідна окремо взятим сервісам. Як приклад можна навести приведення всіх даних до CSV формату з подальшим використанням сформованих файлів у алгоритмах машинного навчання.

Використання спеціалізованих інструментів для роботи з джерелом даних вимагає використання великої кількості обчислювальних ресурсів для виконання задач обробки даних. Крім того, такий підхід передбачає спочатку накопичення певних об’ємів даних і лише потім обробку, що не є ефективним, оскільки більшу частину свого часу ми очікуємо дані і не використовуємо потенціал обчислювальних ресурсів. Дані можуть бути оброблені потоково без очікування заповнення буферів чи настання визначеного часу запуску обробки. Використання потокової обробки дозволяє розбити блок джерела даних на два, як зображено на рисунку 4.4 і оновити загальний погляд на побудову архітектури машинного навчання.

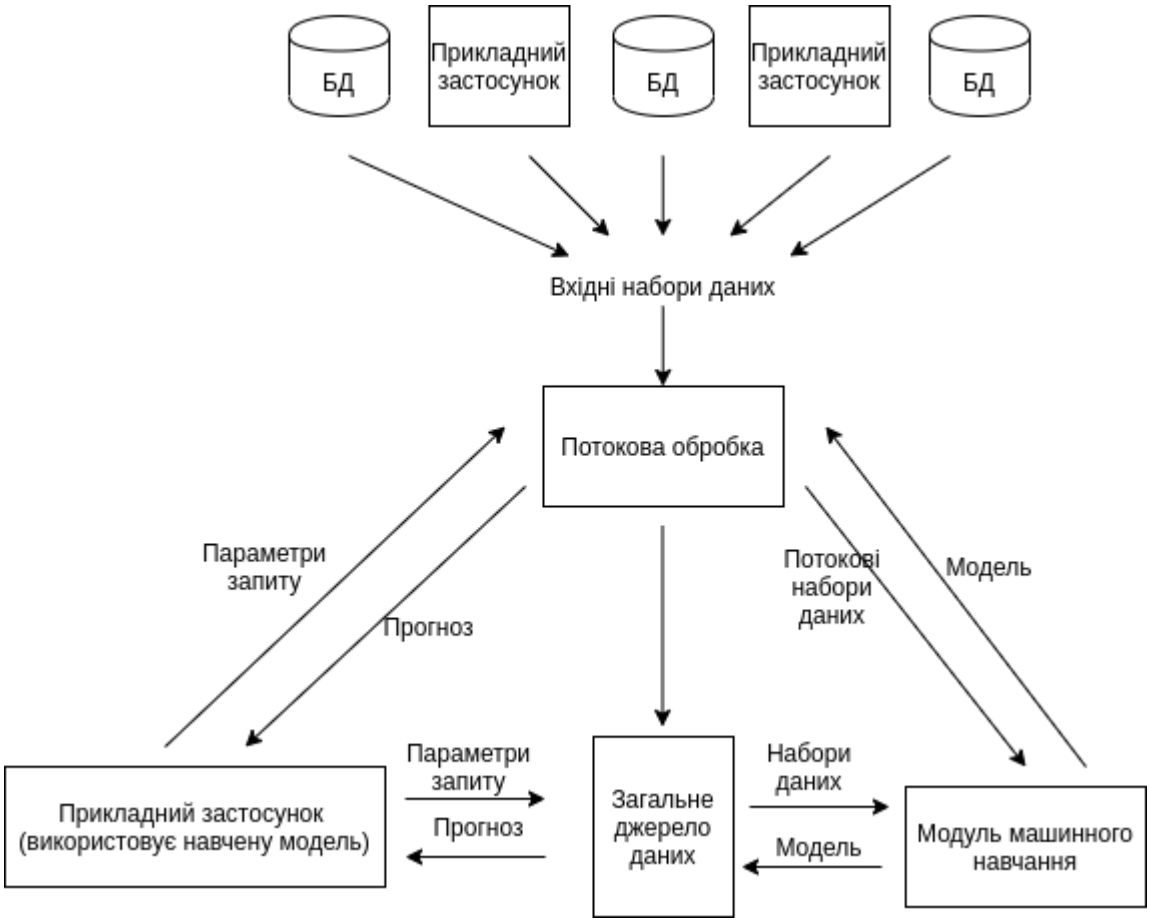

Рисунок 4.4 – Схема процесу машинного навчання з використанням потокової обробки

Сервіси, що відповідають за джерела даних, надсилають повідомлення постійно. Платформа аналітики отримує ці дані або в пакетному, або в реальному часі. Далі використовуються алгоритми машинного навчання для побудови аналітичних моделей. Обробкою такого нескінченного потоку даних займається потокова платформа. Аналітичні моделі розповсюджуються також і неї. Платформа потокової передачі застосовує аналітичні моделі до нових подій для отримання результату (тобто для прогнозування). Результат надсилається споживачеві даних.

Використовуючи такий підхід, ми відокремлюємо навчання моделей від отримання результатів моделі другими сервісами, що є типовим для більшості сучасних систем машинного навчання.

4.2 Архітектура ПЗ

Спираючись на підходи, описані у попередньому розділі, сформуємо архітектуру програмного забезпечення для вирішення поставлених задач (рисунок 4.5). Виділено три головних компоненти системи: користувацький сервіс, що надає нові набори даних, та отримує результати прогнозуванні на основі навченої моделі; файлове сховище, де саме зберігаються набори для навчання та результуючі моделі; обчислювальне середовище, у якому відбувається саме навчання, потокова обробка та донавчання.

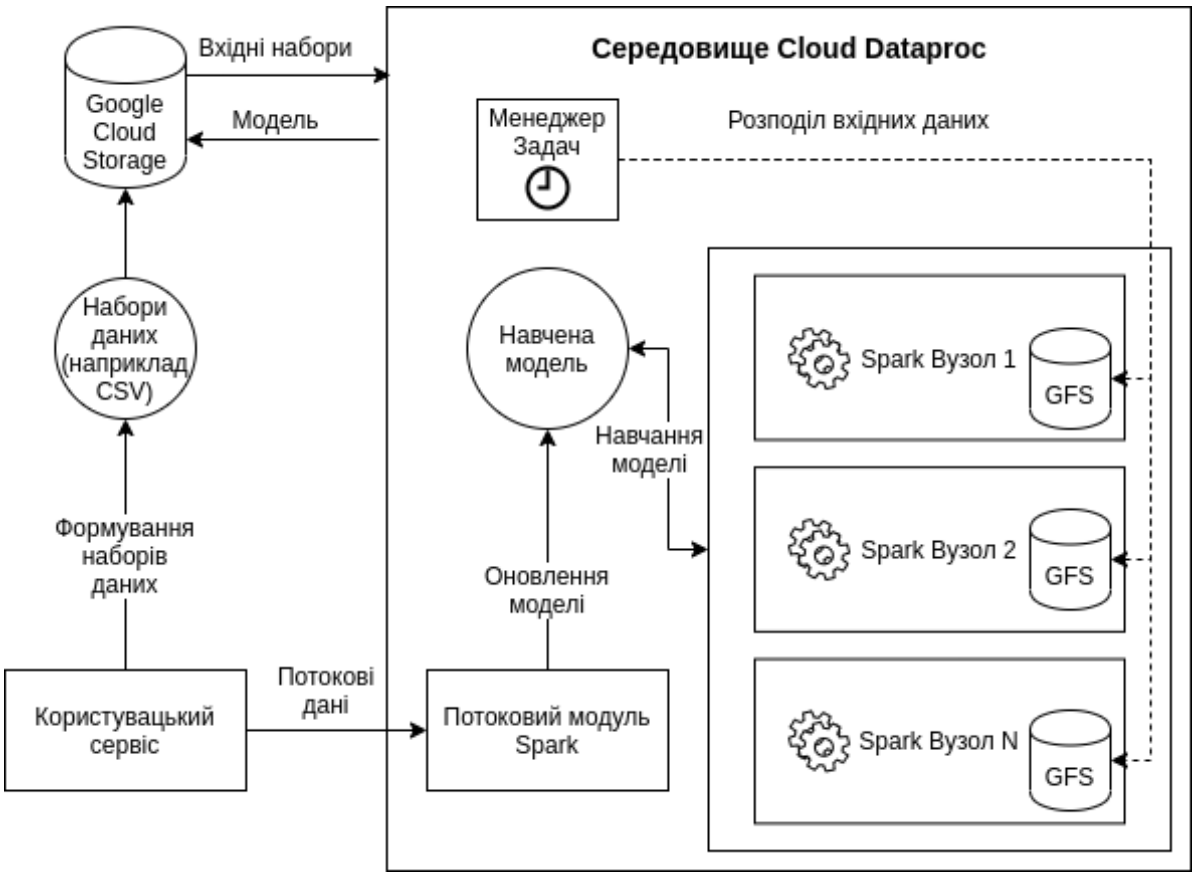

Рисунок 4.5 – Інформаційні потоки процесу машинного навчання

Обчислювальне середовище представляє собою набір з кластеру робочих вузлів Spark, менеджеру задач та модуля потокової обробки даних. Всі дані, що поступають на обробку розподіляються між робочими вузлами та зберігаються у розподіленій файловій системі GFS [38]. Розподілення даних відбувається періодично шляхом завантаження даних з файлового

сховища до обчислювального середовища. Формат, який використовується для наборів даних у файловому сховищі може бути довільним (наприклад текстовий файл, або csv файл). Збереження наборів у GFS використовує спеціалізований формат Parquet, який є вбудованим для інструменту Spark. Загальна схема роботи такого процесу зображена на діаграмі послідовностей на рисунку 4.6.

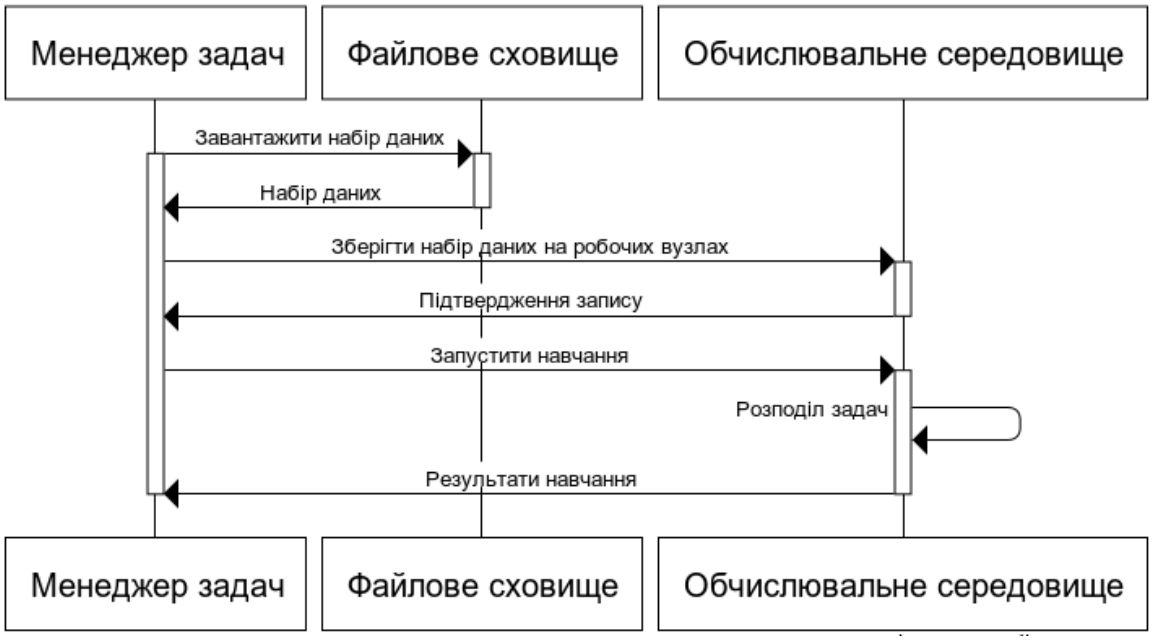

Рисунок 4.6 – Діаграма послідовності обробки навчальних наборів даних

Менеджер задач формує чергу з задач, які розподіляються між робочими вузлами та виконуються у вигляді map-reduce завдань. Результати формують навчену модель, яка може бути оновлена новими даними з модуля потокової обробки Spark. Менеджер чекає накопичення певного об’єму даних або часу для того щоб сформувати набір навчальних даних з потоку і лише потім ставить задачу у чергу на виконання. Навчена модель проходить етап перевірки точності і може оновлювати або заміняти поточну модель як зображено на рисунку 4.7.

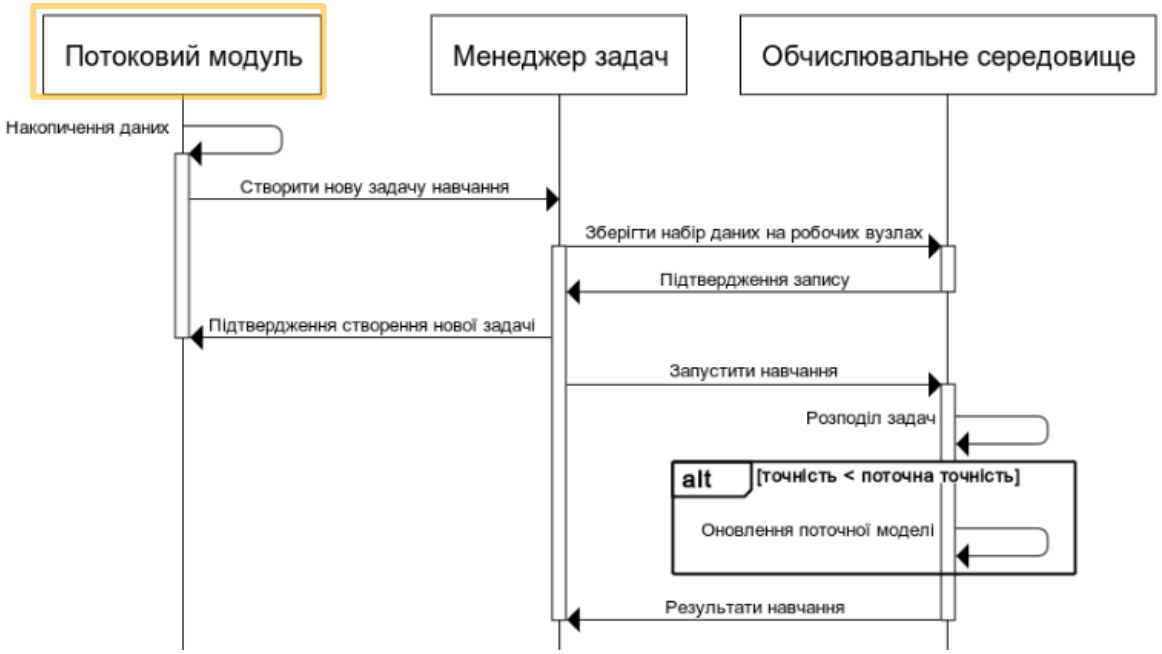

Рисунок 4.7 – Діаграма послідовностей оновлення поточної моделі

4.3 Обробка наборів даних за допомогою Apache Spark

Для обробки наборів даних використовується Apache Spark [1]. Це уніфікований обчислювальний механізм і набір бібліотек для паралельної обробки даних на багатьох вузлах комп'ютерних кластерів. Spark є найбільш активно розвинутим механізмом з відкритим кодом для виконання цього завдання, що робить його стандартним інструментом для будь-якого розробника або вченого, який цікавиться великими даними. Spark підтримує кілька широко використовуваних мов програмування (Python, Java, Scala та R), включає бібліотеки для різноманітних завдань та працює в будь-якому місці від ноутбука до кластеру тисяч серверів. Це полегшує систему, починаючи з масштабної обробки великих даних або неймовірно великого масштабу.

Основним його компонентом є драйвер, який займається зберіганням основної інформації та стану програми, обробкою та виконанням запитів користувача та аналізом і розподілом обчислювальних задач. Крім нього, є певна кількість виконавців, які займаються виконанням обчислень, отриманих від драйвера. За рахунок використання мови Java для створення

програми, рушій Spark можна розгорнути в тій же віртуальній машині, що й інші компоненти програми, що значно спрощує розробку та пришвидшує роботу класифікатора.

Як і в інших системах великих, кластер Spark складається з одного вузла драйвера та кількох робочих вузлів, як показано на рисунку 4.8. Драйвер відповідає за координацію завдань (наприклад, планування та відправлення завдань), в той час як працівники (робочі вузли) відповідають за фактичні розрахунки. Щоб автоматично розпаралелювати обробку даних через кластер у відмовному режимі, Spark пропонує функціональну модель обчислень, паралельну даним. У роботі Spark дані представлені у вигляді стійкого розподіленого набору даних (RDD) [44], який є незмінним набором записів, розділених на кластері, і може бути перетворений лише для отримання нових RDD (тобто копіювання під час запису) через функціональні оператори, як map, filter та reduce.

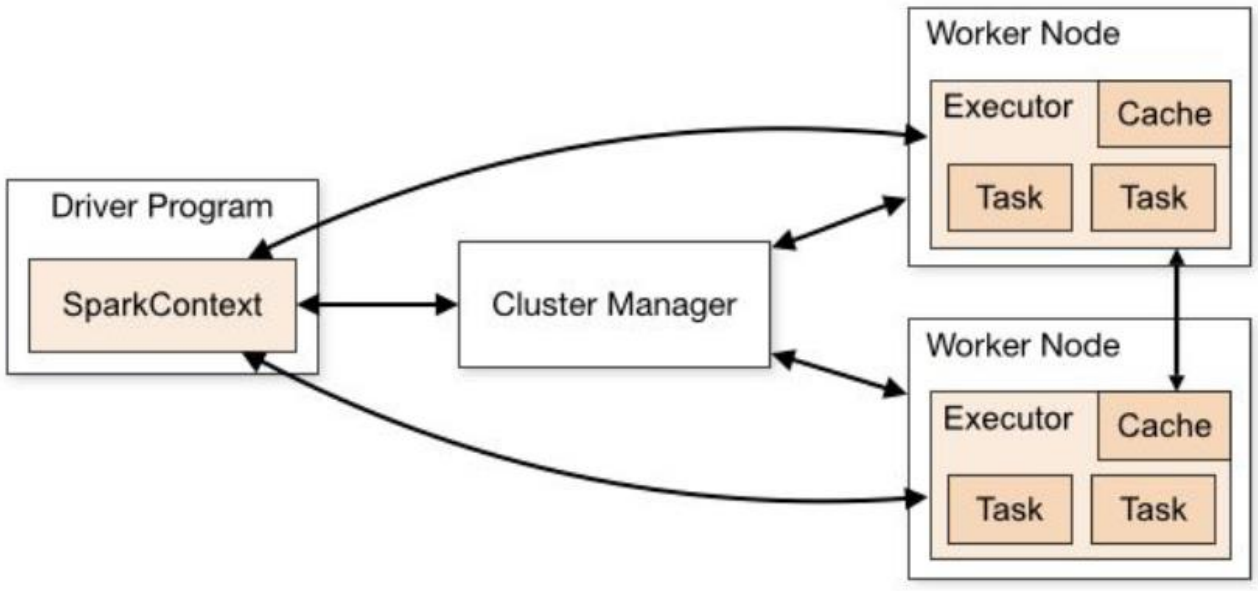

Рисунок 4.8 – Схема розподіленого режиму роботи Apache Spark

4.4 Параметри моделі дерев ізоляцій

Побудова та навчання моделі виконується з використанням класів IsolationForest та IsolationForestModel. Параметрами моделі є:

– numEstimators – кількість дерев в ансамблі;

– maxSamples – кількість елементів (екземплярів), які використовуються для тренування кожного дерева. Якщо це значення знаходиться між 0.0 та 1.0, то воно трактується як частка від загальної кількості. Інакше, то це трактується як кількість;

– contamination – частка аномалій у наборі даних навчання. Якщо значення цього параметру встановлено 0.0, - це прискорить навчання, але всі передбачені результати будуть помилковими. В іншому випадку цей параметр не впливає на модель та результати оцінки;

– contaminationError – помилка при розрахунку порогу, що є необхідним для досягнення зазначеної частки аномалій. За замовчуванням – 0.0, що змушує точно розрахувати поріг. Точний розрахунок повільний і може бути невдалим для великих наборів даних. Якщо є проблеми з точним розрахунком, гарним вибором для цього параметра є 1% від зазначеного значення аномалій;

– maxFeatures – кількість ознак, які використовуються для тренування кожного дерева. Якщо це значення знаходиться між 0.0 та 1.0, то воно трактується як частка від загальної кількості. Інакше, це трактується як кількість;

– randomSeed – початкове значення для генератора випадкових чисел;

– featuresCol – назва стовпця для вектору ознак. Цей стовпець повинен існувати у вхідному DataFrame після проведення навчання;

– predictionCol – назва стовпця для результатів прогнозування (передбачення). Цей стовпець додається до вхідного DataFrame після проведення навчання;

– scoreCol – показник аномальності. Цей стовпець додається до вхідного DataFrame після проведення навчання.

4.5 Розподілене обчислення у середовищі Dataproc

Виконувати навчання можна як на одному вузлі локально, так і на декількох розподіллено. Оскільки зараз широкої популярності набирає обчислення з використанням хмарних рішень, то інструменти для розподілення стають більш доступними. Одним з таких є Cloud Dataproc, що надається як один з інструментів у комплекті Google Cloud [61].

Cloud Dataproc [62] – це платформа для керування Apache Spark та Apache Hadoop, яка дозволяє вам скористатися цими інструментами для обробки великих наборів даних, потокової передачі та машинного навчання. Автоматизований процес Dataproc допомагає швидко створювати кластери на довільну кількість робочих вузлів, легко керувати ними з одного місця та економити витрати як грошові, так і обчислювальні за рахунок відключення тих чи інших вузлів, які зараз не використовуються. Маючі менше проблем та менше витрат на налаштування та адміністрування інфраструктури, ви маєте більше часу та можливостей зосередитися на вирішенні задач по обробці даних та отримання корисного результату з цього.

Налаштування проекту є досить простим оскільки вимагає лише створення облікового запису у Google Cloud та унікальної назви нашого застосунку (рисунок 4.9 та рисунок 4.10). Для нас це ще буде і назвою кластеру, до якого ми маємо підключатися, надсилати завдання на виконання і виконувати машинне навчання.

Рисунок 4.9 – Створення нового проекту у середовищі Google Cloud

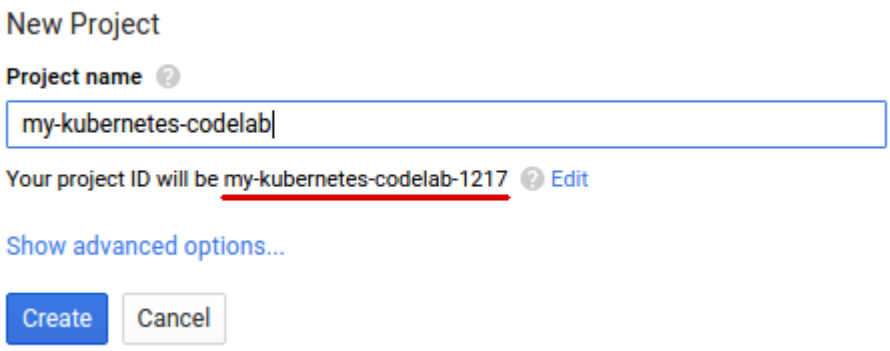

Рисунок 4.10 – Налаштування нового проекту

Кожен вузол у системі, не зважаючи на те чи це головний, чи робочий вузол, запускається у хмарному середовищі як окрема віртуальна машина. Такий підхід дозволяє нам не задумуватися над деталями підтримки фізичних пристроїв та дисків, а делегувати цей процес хмарному середовищі. Сам же Dataproc використовує ці обчислювальні ресурси задля контролю виконання задач на рівні Apache Spark. Такий підхід до делегування дозволяє розділити свої відповідальності між підтримкою фізичних пристроїв, контроль розподілених обчислень, та власне машинного навчання прикладних моделей для вирішення конкретних задач.

Контроль обчислювальних ресурсів виконується за допомогою Google Compute Engine – набору сервісів та послуг для розгортання власних застосунків на фізичних пристроях з можливістю віртуалізації окремих процесів. Знайти та підключити такий набір інструментів можна через Google Cloud Console як показано на рисунку 4.11 та рисунку 4.12.

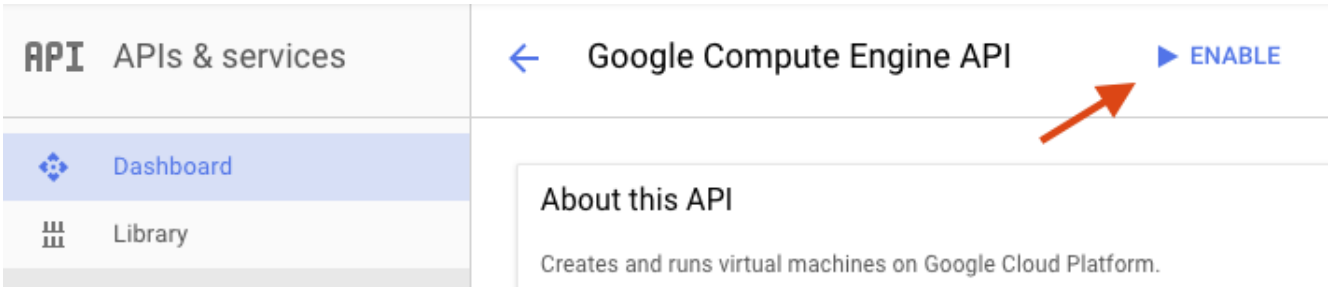

Рисунок 4.11 – Підключення інструментів Google Compute Engine

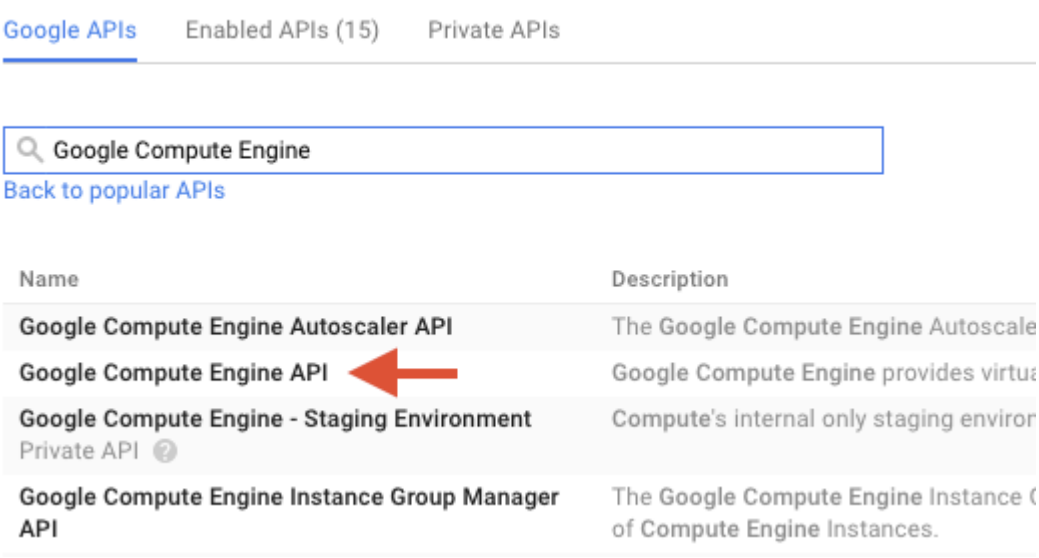

Рисунок 4.12 – Вибір Google Compute Engine інструментів з загального списку

Створення кластеру вже виконується у межах Dataproc шляхом натискання відповідних кнопок та вказання параметрів кластеру. У нашому випадку це інформація про розташування кластеру та тип машин (потужність, та об'єми пам'яті), що мають використовуватися у кластері. Результатом виконання таких дій є створений кластер, який відображений у загальному списку.

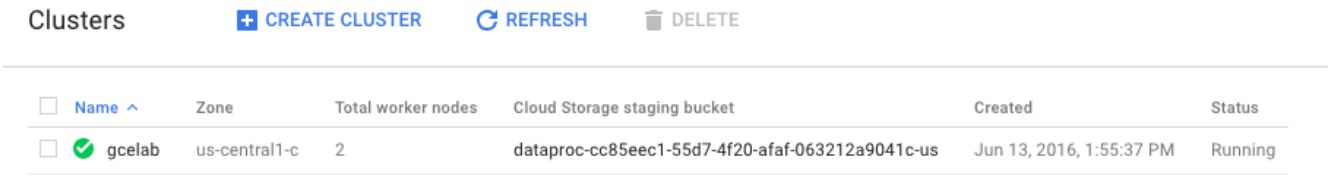

Рисунок 4.13 – Створений кластер з двома робочими вузлами

Виконання задач у середовищі Spark відбувається за допомогою вбудованого інструменту командного рядка spark-submit. Цей інструмент дозволяє надсилати код програми до кластеру і запускати його виконання (рисунок 4.14). Після того, як код задачі потрапляє до кластеру, система з

декількох вузлів намагається його виконати і не завершить своє виконання до тих пір, поки завдання не буде виконане успішно, або завершиться з помилкою. Хід виконання задачі контролюється автоматично за допомогою менеджерів кластеру, таких як Mesos та YARN. spark-submit пропонує кілька елементів керування, за допомогою яких ви можете вказати необхідні ресурси для вашої програми, а також те, як це слід запускати та аргументи командного рядка. У середовищі Dataproc такий інструмент як spark-submit є вбудованим і дозволяє налаштовувати параметри задачі з використанням зручного інтерфейсу.

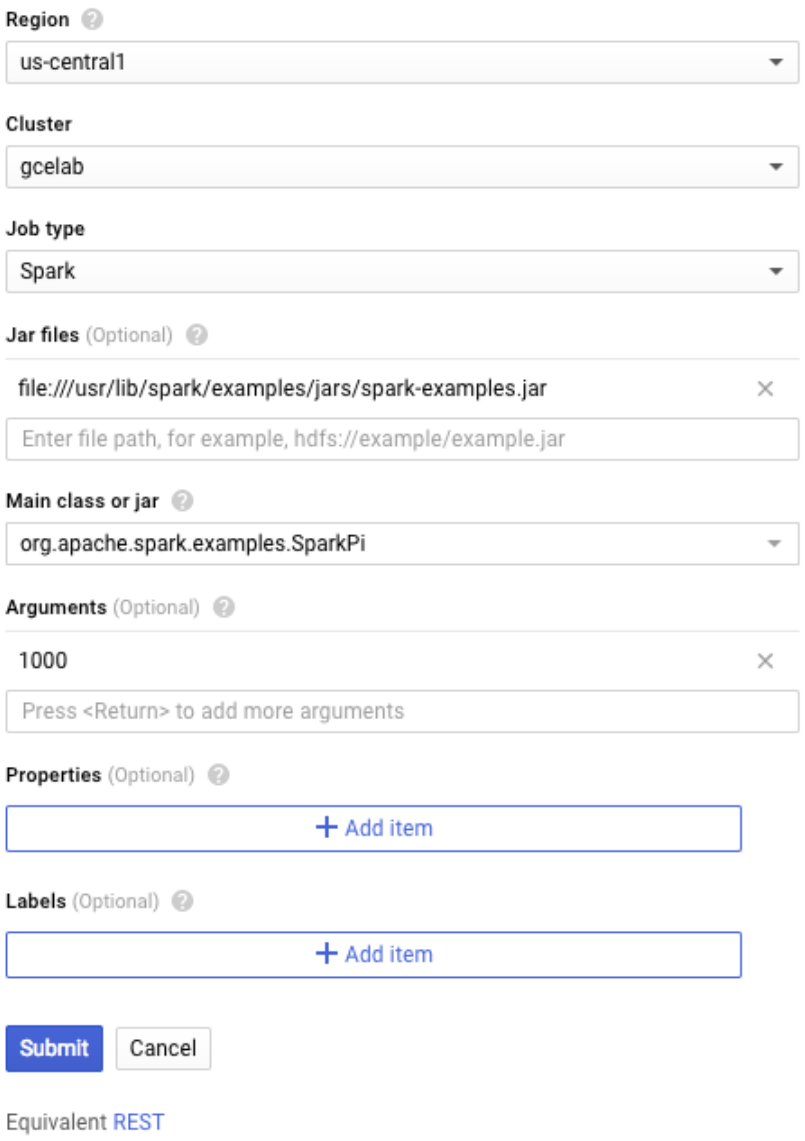

Рисунок 4.14 – Налаштування параметрів виконання для задачі Spark

Результати виконання задач можна побачити у окремому розділі як зображено на рисунку 4.15.

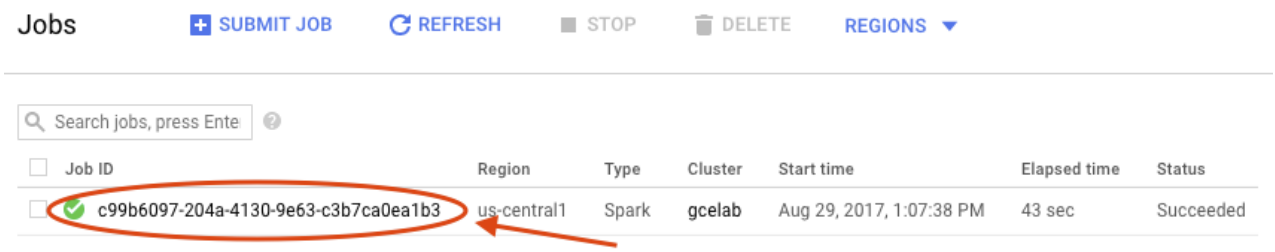

Рисунок 4.15 – Результати виконання задачі Spark

Висновки до розділу

У розділі була розглянута програмна реалізація підходу до розподіленого машинного навчання з використанням потоків даних, було розглянути загальні підходи, спроектована власне архітектурне рішення.

Обробка наборів даних виконується за допомогою рушія Apache Spark у хмарному середовищі. Підтримка розподіленої інфраструктури делегована на сервіс Cloud Dataproc, що дозволяє абстрагуватися від фізичних вузлів у розподіленій системі та сконцентрувати увагу на вирішенні задач, пов’язаних з обробкою великих наборів даних та потоків.

Виконання задач навчання на вузлах розподіленої системи доступне як через графічний інтерфейс Dataproc, так і через інструмент командного рядка spark-submit. Результати навчання для подальшого аналізу доступні як результати виконаних завдань у кластері та як збережені моделі на жорстких дисках.

5 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА МЕТОДУ РОЗПОДІЛЕНОГО МАШИННОГО НАВЧАННЯ

Розроблений в цьому дисертаційному дослідженні метод розподіленого машинного навчання та програмне забезпечення, що його реалізує, використовують розподілене середовище для виконання навчання. Для перевірки розробленого методу було проведено порівняння такого підходу з виконанням навчання на одному робочому вузлі, тобто без розподілення, з різною кількість робочих вузлів у системі та проведені відповідні заміри показників у залежності від розміру навчальних наборів, налаштувань навчання.

5.1 Опис набору даних

Оригінальна стаття, що описує метод дерев та лісу ізоляцій для пошуку аномалій, використовує відкриті набори даних. Одним з таких є набір shuttle (statlog), який містить інформацію про авіа рейси. Цей набір даних містить 500 тисяч елементів і 9 різних класів для класифікації. Тут об’єднані як дані для навчання, так і данні для тестування. Дані є наперед розміченими з загальним значенням аномалій 10%. Найменші п’ять класів, тобто класи з номерами 2, 3, 5, 6, 7, поєднуються, щоб утворювати клас аномалій, тоді як клас 1 утворює правильний (не аномальний) клас. Дані для 4 класу ігноруються.

Вхідні набори даних представляються як CSV файл, де кожен рядок відповідає конкретному екземпляру або вектору ознак. Перші дев’ять значень у векторі є фактичними та реальними ознаками, останнє значення є ознакою аномальності і розмічене вручну. Якщо 1.0, то екземпляр аномальний, якщо 0.0 – не аномальний.

5.2 Опис параметрів моделі

Оскільки більша частина параметрів моделі залежить від конкретної реалізації того, чи іншого середовища виконання та інструменту, за допомогою якого запускається навчання, наведемо лише ті параметри моделі, які впливають на результат обчислень і можуть бути скореговані незалежно від використовуваних підходів.

Такими параметрами є:

– numEstimators – встановлює кількість дерев в ансамблі, чим більше дерев – тим точніше результат;

– maxSamples – встановлює кількість елементів (екземплярів), які використовуються для тренування кожного дерева, що дозволяє використовувати не всі екземпляри з набору, а лише певну фіксовану частину або відсоток, що дозволяє формувати унікальні та незалежні групи;

– contamination – частка аномалій у наборі навчальних даних; якщо данні не розміченні, то параметр може бути підібраний емпірично, з урахуванням предметної області та бажаного відсотку аномалій.

5.3 Експеримент зі швидкістю навчання

В залежності від кількості робочих вузлів наявних у системі та розмірів набору даних, швидкість навчання може різнитися. У ході виконання дослідження, було проведено експерименти на наборах даних різної розмірності та з різними вхідними параметрами. Результати наведені у вигляді таблиць та графіків.

Параметри моделі ізольованого лісу:

– numEstimators = 100;

– maxSamples = 256;

– contamination = 0.1.

На рисунку 5.1 наведений графік швидкості навчання у залежності від кількості екземплярів у наборі даних для різної кількості робочих вузлів у кластері.

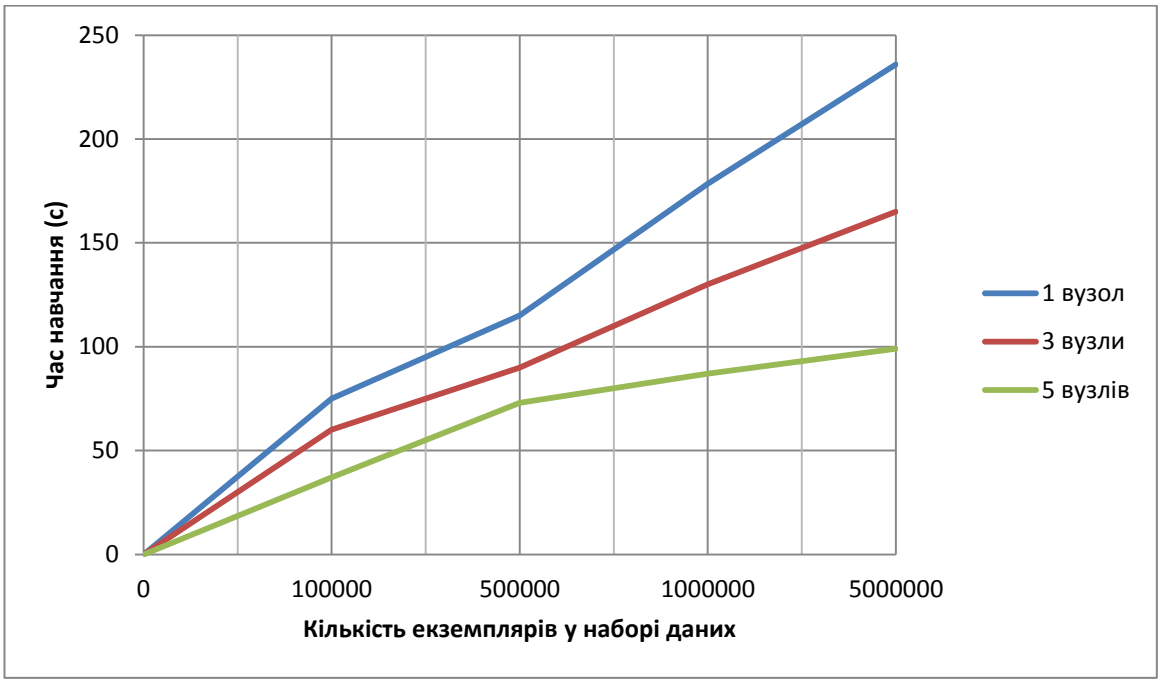

Рисунок 5.1 – Залежність швидкості навчання від кількості екземплярів у наборі даних та кількості робочих вузлів у системі

Результати експерименту дають привід зроби висновок, що зі збільшенням кількості робочих вузлів у системі зменшується і час навчання моделі. Проте, варто відзначити, що така закономірність може бути застосована лише тоді, коли набори даних великі (мають більше 100 000 екземплярів). Такий результат можливо поясними тим, що на малих наборах даних приріст не є суттєвим, оскільки більшу частину процесорного часу займає розбиття задач на підзадачі, передача даних мережею, а не фактичне виконання обчислень. Можливим вирішенням такої проблеми є підготовка розбитих наборів даних на окремих робочих вузлах системи.

5.4 Експеримент з донавчанням моделі

Донавчання моделі виконується задля покращення результатів основного навчання та задля оновлення актуальності моделі у результаті її

деградації через явище дрейфу. У ході дослідження використовується потоковий підхід до вирішення такої проблеми.

Потік даних складається з невеликих наборів даних, кожен з яких містить 100 000 екземплярів. Первинне навчання відбувається на такому ж наборі даних а виконує роль відправної точки для всього потоку. У ході експерименту, робота з потоком розбита на кроки, де кожен крок відповідає новому набору даних, який надходить на всіх. Кожен раз коли виконується прогнозування результатів підраховується точність отриманого результату і порівнюється з очікуваною. Варто відзначити, щоразу на вхід подаються дані, що поступово збільшують відхилення від початкового навчання, тобто імітується деградація моделі. Через декілька таких кроків відбувається перенавчання на поточному наборі даних і процес повторюється знову. На рисунку 5.2 зображений графік залежності точності моделі від ітерацій потокової обробки.

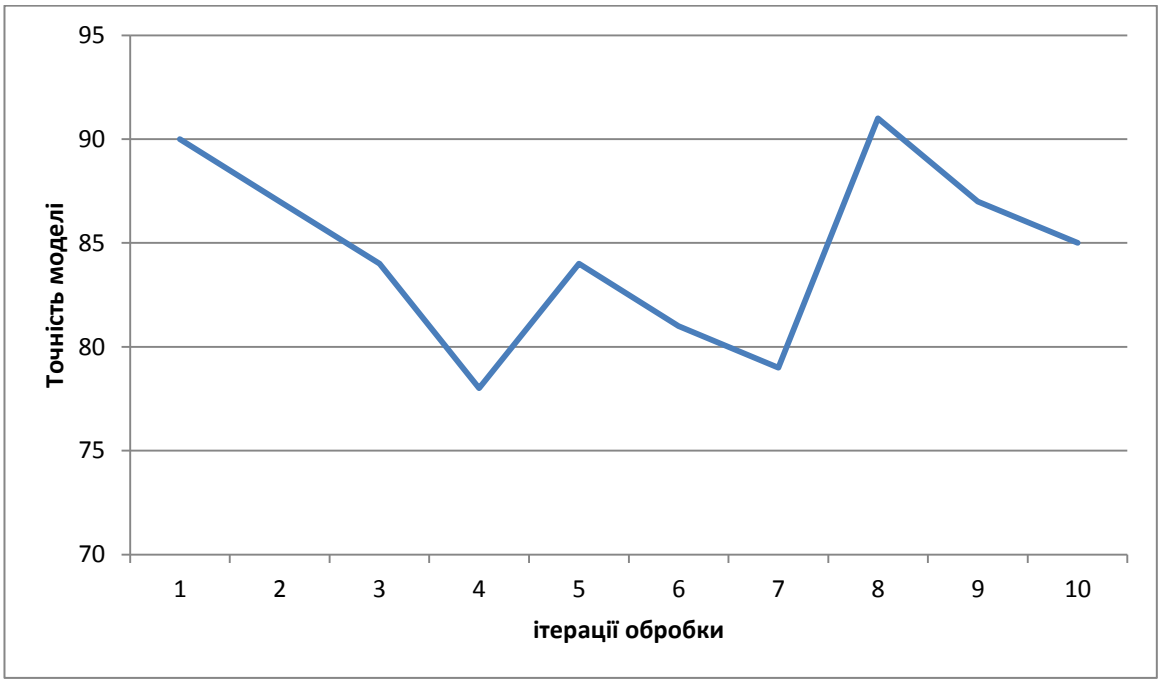

Рисунок 5.2 – Залежність точності моделі від ітерацій потокової обробки

Як результат, ми бачимо, що проблема дрейфу може бути вирішена без повного перенавчання моделі і у режимі реального часу шляхом обробки

поточного набору даних. Відсутність поступової деградації моделі у більшу чи меншу сторону неможлива, оскільки вхідні дані мають властивість змінюватися у залежності від часу.

Висновки до розділу

В цьому розділі наведено результати виконання розподіленого машинного навчання на декількох обчислювальних вузлах системи. Найбільшої ефективності досягнуто за рахунок виконання машинного навчання на великих наборах даних з більшою кількістю обчислювальних вузлів, а саме 5. Проблема дрейфу і збільшення точності під час деградації моделі вирішується за допомогою введення потокової обробки та донавчання головної моделі, що не дозволяє опускатися точності моделі нижче 78% протягом часу виконання прогнозування.

ВИСНОВКИ

У цій роботі розроблено метод розподіленого машинного навчання з використанням алгоритму лісу ізоляцій на прикладі вирішення задачі пошуку аномалій.

У першому розділі проведено загальний огляд методів та підходів до розподіленого машинного навчання: особливості обчислень виразів лінійної алгебри, використання апаратного прискорення, системи обміну повідомленнями, паралелізм рівня даних, паралелізм рівня моделі, вертикальне та горизонтальне масштабування. Розглянуті технології, які можуть бути застосовані для розподілення навчання: розподілені файлові системи, топології формування взаємодії між вузлами системи, підходи до проведення обчислень, алгоритми машинного навчання. Встановлено, що існуючі рішення машинного навчання використовують вертикальне масштабування і що є перспективи використання горизонтального масштабування з залученням більшої кількості обчислювальних вузлів. На основі проведеного огляду сформовано задачі, які необхідно вирішити у рамках цього дослідження.

У другому розділі розглянутий підхід до розподіленого машинного навчання на прикладі пошуку аномалій. Описана задача пошуку аномалій, підходи та сфери її застосування. Задля вирішення задачі, запропонований алгоритм з використанням дерев та лісу ізоляцій, для якого виділено особливості та можливості до розподілу його виконання на декількох робочих вузлах.

У розділі розробки методу розподіленого машинного навчання описано метод навчання з використання паралелізму рівня даних, формуванням менших матриць ознак, виокремленням окремих векторів. Описано явище деградації навченої моделі протягом часу. Розроблений метод передбачає використання рушія Apache Spark, який відповідає за виконання розподіленого навчання, збереження даних у розподіленій файловій системі і підтримує потокову обробку даних.

У розділі програмної реалізації розподіленого машинного навчання описано підходи до побудови інформаційних потоків машинного навчання. Як результат, сформована загальна архітектура та описані основні компоненти розподіленої системи., що підтримує використання розподіленого обчислювального середовища і розподіленої файлової системи задля виконання своїх задач.

У розділі експериментальної перевірки були описані результати перевірки програмного забезпечення, що базується на розробленому у третьому розділі методі. Визначено напрямки подальших досліджень: залучення потокової обробки, збільшення кількості обчислювальних вузлів, паралельне навчання декількох моделей.

Наукова новизна роботи полягає у створенні методу розподіленого навчання на основі використання розподілених даних, обчислювальних ресурсів та залучення потокової обробки даних. Метод базується на поєднанні традиційних алгоритмів та підходів до машинного навчання з розподіленими можливостями сучасних систем. Наступним кроком дослідження є модифікація методу для можливості розширення її використання на інших обчислювальних рушіях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Spark: Cluster computing with working sets. / [M. Zaharia, M. Chowdhury, M. Franklin та ін.]. // HotCloud. – 2010. – №10. – С. 95.

2. Mllib: Machine learning in apache spark. / [X. Meng, J. Bradley, B. Yavuz та ін.]. // The Journal of Machine Learning Research. – 2016. – №17. – С. 1235–1241.

3. Chollet F. Keras. [Електронний ресурс] / François Chollet. – 2015. – Режим доступу до ресурсу: https://keras.io/.

4. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Електронний ресурс] / [M. Abadi, A. Agarwal, P. Barham та ін.]. – 2015. – Режим доступу до ресурсу: https://www.tensorflow.org/.

5. Basic linear algebra subprograms for Fortran usage. / [C. Lawson, R. Hanson, D. Kincaid та ін.]. // ACM Transactions on Mathematical Software (TOMS). – 1979. – №5. – С. 308–323.

6. Using MPI: portable parallel programming with the message-passing interface. / [W. D. Gropp, W. Gropp, E. Lusk та ін.]. // MIT press. – 1999. – №1.

7. Raina R. Large-scale deep unsupervised learning using graphics processors. In Proceedings of the 26th annual international conference on machine learning / R. Raina, A. Madhavan, A. Y Ng. // ACM. – 2009. – С. 873–880.

8. Theano: A CPU and GPU math compiler in Python / [J. Bergstra, O. Breuleux, F. Bastien та ін.]. // In Proc.9th Python in Science Conf. – 2010. – №1.

9. Caffe: Convolutional architecture for fast feature embedding / [Y. Jia, E. Shelhamer, J. Donahue та ін.]. // In Proceedings of the 22nd ACMinternational conference on Multimedia. – 2014. – С. 675–678.

10. Nvidia Tesla V100. [Електронний ресурс] // NVIDIA Corporation. – 2017. – Режим доступу до ресурсу: https://www.nvidia.com/en-us/data-center/tesla-v100/.

11. Flynn M. Some computer organizations and their effectiveness. / Michael Flynn. // IEEE transactions on computers. – 1972. – №100. – С. 948–960.

12. Han T. Reducing branch divergence in GPU programs. / T. Han, T. Abdelrahman. // ACM International Conference Proceeding Series. – 2011. – №3.

13. Meuth R. GPUs surpass computers at repetitive calculations. / Ryan Meuth. // IEEE Potentials 26. – 2007. – С. 12–23.

14. Metz C. Big bets on AI open a new frontier for chip start-ups, too. / Cade Metz. // The New York Times. – 2018. – №14.

15. Smith M. Application-specific integrated circuits / Michael Smith. // Addison-Wesley Reading, MA. – 1997. – №7.

16. Sato K. An in-depth look at Google's first Tensor Processing Unit (TPU) / K. Sato, C. Young, D. Patterson. // Google Cloud Big Data and Machine Learning Blog. – 2017. – №12.

17. In-datacenter performance analysis of a tensor processing unit. / [N. Jouppi, C. Young, N. Patil та ін.]. // ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). – 2017. – С. 1–12.

18. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. / [T. Chen, Z. Du, N. Sun та ін.]. // ACM Sigplan Notices 49. – 2014. – №4. – С. 269–284.

19. Neuffow: A runtime reconfigurable dataflow processor for vision. / [C. Farabet, B. Martini, B. Corda та ін.]. // Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on. IEEE. – 2011. – С. 109–116.

20. Le Q. Building high-level features using large scale unsupervised learning. / Quoc Le. // Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE. – 2013. – С. 8595–8598.

21. Sermanet P. Traffic sign recognition with multi-scale convolutional networks. / P. Sermanet, Y. LeCun. // Neural Networks (IJCNN), The 2011 International Joint Conference on. IEEE. – 2011. – С. 2809–2813.

22. Improving neural networks by preventing co-adaptation of feature detectors / [G. Hinton, N. Srivastava, A. Krizhevsky та ін.]. – 2012.

23. Reinders J. AVX-512 instructions / James Reinders. // Intel Corporation. – 2013.

24. Olofsson A. Kickstarting high-performance energy-efficient manycore architectures with epiphany. / A. Olofsson, T. Nordström, Z. Ul-Abdin. // arXiv. – 2014. – №1413.

25. Olofsson A. Epiphany-V: A 1024 processor 64-bit RISC System-On-Chip. / Andreas Olofsson. – 2016.

26. E64G401 EPIPHANY 64-CORE MICROPROCESSOR Datasheet. [Електронний ресурс] // Adapteva, Inc.. – 2017. – Режим доступу до ресурсу: http://www.adapteva.com/docs/e64g401_datasheet.pdf..

27. Introducing Amazon EC2 P2 Instances, the largest GPU-Powered virtual machine in the cloud. [Електронний ресурс] // Amazon Web Services.. – 2016. – Режим доступу до ресурсу: https://aws.amazon.com/about-aws/whats-new/2016/09/introducing-amazon-ec2-p2-instances-the-largestgpu-powered-virtual-machine-in-the-cloud/.

28. Coulouris G. Distributed systems: concepts and design. / G. Coulouris, J. Dollimore, T. Kindberg. // Pearson Education.. – 2005.

29. A survey of rollbackrecovery protocols in message-passing systems. / E.Elnozahy, L. Alvisi, Y. Wang, D. Johnson. // ACM Computing Surveys (CSUR) 34. – 2002. – №3. – С. 375–408.

30. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. / [M. Ferdman, A. Adileh, O. Kocberber та ін.]. // ACM Sigplan Notices. – 2012. – №47. – С. 37–48.

31. Haußmann E. Accelerating I/O bound deep learning on shared storage. [Електронний ресурс] / Elmar Haußmann. – 2018. – Режим доступу до ресурсу: https://blog.riseml.com/acceleratingio-bound-deep-learning-e0e3f095fd0..

32. Barroso L. Web search for a planet: The Google cluster architecture. / L. Barroso, J. Dean, U. Hölzle. // IEEE micro. – 2003. – №2. – С. 22–28.

33. Genomic big data hitting the storage bottleneck. / [L. Papageorgiou, P. Eleni, S. Raftopoulou та ін.]. // EMBnet. – 2018. – №24.

34. Distributed algorithms for topic models. / D.Newman, A. Asuncion, P. Smyth, M. Welling. // Journal of Machine Learning Research. – 2009. – №10. – С. 1801–1828.

35. Richtárik P. Distributed coordinate descent method for learning with big data. / P. Richtárik, M. Takáč. // The Journal of Machine Learning Research 17.. – 2016. – №1. – С. 2657–2681.

36. Peteiro-Barral D. A survey of methods for distributed machine learning. / D. Peteiro-Barral, B. Guijarro-Berdiñas. // Progress in Artificial Intelligence 2. – 2013. – №1. – С. 1–11.

37. Strategies and principles of distributed machine learning on big data. / E.Xing, Q. Ho, P. Xie, D. Wei. // Engineering 2. – 2016. – №2. – С. 179–195.

38. Ghemawat S. The Google File System. / S. Ghemawat, H. Gobioff, S. Leung. // ACM Symposium on Operating Systems Principles. – 2003.

39. Big data analysis using Apache Hadoop. / [J. Nandimath, E. Banerjee, A. Patil та ін.]. // IEEE 14th International Conference on Information Reuse & Integration (IRI). – 2013. – С. 700–703.

40. The hadoop distributed !le system. / K.Shvachko, H. Kuang, S. Radia, R. Chansler. // Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on.. – 2010. – С. 1–10.

41. Dean J. MapReduce: Simplified Data Processing on Large Clusters. / J. Dean, S. Ghemawat. // Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (OSDI'04). USENIX Association, Berkeley, CA, USA. – 2004. – С. 10–10.

42. Lammel R. Google's MapReduce programming model. / Ralf Lammel. // Revisited. Science of computer programming 70. – 2008. – №1.

43. Shanahan J. Large scale distributed data science using apache spark. / J. Shanahan, L. Dai. // Proceedings of the 21th ACM SIGKDD international

conference on knowledge discovery and data mining. ACM,. – 2015. – С. 2323–2324.

44. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. / [M. Zaharia, M. Chowdhury, T. Das та ін.]. // Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association. – 2012. – С. 2–2..

45. TR-Spark: Transient Computing for Big Data Analytics. / [Y. Yan, Y. Gao, Y. Chen та ін.]. // Proceedings of the Seventh ACM Symposium on Cloud Computing (SoCC '16). ACM, New York, NY, USA. – 2016. – С. 484–496.

46. Blei D. Latent dirichlet allocation. / D. Blei, A. Ng, M. Jordan. // Journal of machine Learning research. – 2003. – №3. – С. 993–1022.

47. Opitz D. Popular ensemble methods: An empirical study. / D. Opitz, R. Maclin. – 1999.

48. Automatic differentiation in PyTorch. / [A. Paszke, S. Gross, S. Chintala та ін.]. – 2017.

49. Google Cloud TPU. [Електронний ресурс] // Google. – 2017. – Режим доступу до ресурсу: https://cloud.google.com/tpu.

50. Microsoft Azure Machine Learning. [Електронний ресурс] // Microsoft. – 2018. – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/overview/machine-learning/.

51. Accelerating deep convolutional neural networks using specialized hardware. / [K. Ovtcharov, O. Ruwase, J. Kim та ін.]. // Microsoft Research Whitepaper 2. – 2015. – №11.

52. Amazon SageMaker. [Електронний ресурс] // Amazon Web Services. – 2018. – Режим доступу до ресурсу: https://aws.amazon.com/sagemaker/developer-resources/.

53. IBM Watson Machine Learning. [Електронний ресурс] // IBM Cloud. – 2018. – Режим доступу до ресурсу: https://www.ibm.com/cloud/machine-learning.

54. Koren Y. Matrix Factorization Techniques for Recommender Systems. / Y. Koren, R. Bell, C. Volinsky. // Computer 42. – 2009. – №8. – С. 30–37.

55. Gönen M. Predicting drug–target interactions from chemical and genomic kernels using Bayesian matrix factorization. / Mehmet Gönen. // Bioinformatics 28. – 2012. – №18. – С. 2304–2310.

56. Macau: Scalable Bayesian factorization with high-dimensional side information using MCMC. / [J. Simm, A. Arany, P. Zakeri та ін.]. // In 2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP).. – 2017.

57. Ю.О. Олійник, О.Є. Афанасьєва, Г.Д.Аршакян Підхід до виявлення аномалій в потоках текстових даних. «Системні технології» 2 (127) 2020 – С.126-139. DOI: https://doi.org/10.34185/1562-9945-2-127-2020-10

58. Tomashevskii, V. M., Oliynik, Y. O., Yaskov, V. V., Romanchuk, V. M. (2018). Realtime text stream anomalies analysis system. Вісник Херсонського національного технічного університету, (3 (1)), 361-365.

59. Liu F. Isolation forests. / F. Liu, K. Ting, Z. Zhou. // In Proceedings of International Conference on Data Mining. – 2008.

60. Heng W. Concept Drift Detection for Streaming Data / W. Heng, A. Zubin. // in the International Joint Conference of Neural Networks. – 2015.

61. Google Cloud [Електронний ресурс] // Google – Режим доступу до ресурсу: https://cloud.google.com/.

62. Cloud Dataproc [Електронний ресурс] // Google – Режим доступу до ресурсу: https://cloud.google.com/dataproc.

ДОДАТОК А – ПРОГРАМНИЙ КОД

IsolationForest.scala

```
package ua.kpi.master.isolationforest


import ua.kpi.master.isolationforest.Utils.{DataPoint, OutlierScore}
import org.apache.spark.internal.Logging
import org.apache.spark.ml.linalg.SQLDataTypes.VectorType
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.ml.util.{DefaultParamsReadable, DefaultParamsWritable, Identifiable}
import org.apache.spark.ml.Estimator
import org.apache.spark.sql.types.StructType
import org.apache.spark.sql.Dataset
import org.apache.spark.{HashPartitioner, TaskContext}


class IsolationForest(override val uid: String) extends Estimator[IsolationForestModel]
  with IsolationForestParams with DefaultParamsWritable with Logging {


  def this() = this(Identifiable.randomUID("isolation-forest"))


  override def copy(extra: ParamMap): IsolationForest = {


    copyValues(new IsolationForest(uid), extra)
  }



  override def fit(data: Dataset[_]): IsolationForestModel = {


    import data.sparkSession.implicits._


    transformSchema(data.schema, logging = true)
    val df = data.toDF()
    val dataset = df.map(row =>
      DataPoint(row.getAs[Vector]($(featuresCol)).toArray.map(x => x.toFloat)))
```

```
val totalNumFeatures = dataset.head.features.length
val numFeatures = if ($(maxFeatures) > 1.0) {
  math.floor($(maxFeatures)).toInt
} else {
  math.floor($(maxFeatures) * totalNumFeatures).toInt
}

val totalNumSamples = dataset.count()
val numSamples = if ($(maxSamples) > 1.0) {
  math.floor($(maxSamples)).toInt
} else {
  math.floor($(maxSamples) * totalNumSamples).toInt
}

val nSigma = 7.0
val targetNumSamples = numSamples.toDouble + nSigma * math.sqrt(numSamples.toDouble)
val sampledRdd = BaggedPoint
  .convertToBaggedRDD(dataset.rdd, sampleFraction, $(numEstimators), $(bootstrap), $(randomSeed))
val flattenedSampledRdd = BaggedPoint.flattenBaggedRDD(sampledRdd, $(randomSeed) + 1)
val repartitionedFlattenedSampledRdd = flattenedSampledRdd
  .partitionBy(new HashPartitioner($(numEstimators)))
val repartitionedFlattenedSampledDataset = repartitionedFlattenedSampledRdd
  .mapPartitions(x => x.map(y => y._2), preservesPartitioning = true)
  .toDS
logInfo(s"Training ${$(numEstimators)} isolation trees using" +
  s" ${repartitionedFlattenedSampledDataset.rdd.getNumPartitions} partitions.")

implicit val isolationTreeEncoder = org.apache.spark.sql.Encoders.kryo[IsolationTree]
val isolationTrees = repartitionedFlattenedSampledDataset.mapPartitions( x => {
  val seed = $(randomSeed) + TaskContext.get.partitionId() + 2
  val rnd = new scala.util.Random(seed)

  val dataForTree = rnd.shuffle(x.toSeq).slice(0, numSamples).toArray
  if (dataForTree.length != numSamples)
```

```
        logWarning(s"Isolation tree with random seed ${seed} is trained using" +
          s" ${dataForTree.length} data points instead of user specified ${numSamples}")

      val featureIndices = rnd.shuffle(0 to dataForTree.head.features.length - 1).toArray
        .take(numFeatures).sorted
      if (featureIndices.length != numFeatures)
        logWarning(s"Isolation tree with random seed ${seed} is trained using" +
          s" ${featureIndices.length} features instead of user specified ${numFeatures}")

      Iterator(IsolationTree
        .fit(dataForTree, $(randomSeed) + $(numEstimators) + TaskContext.get.partitionId() + 2,
featureIndices))
    }).collect()

    val isolationForestModel = copyValues(
      new IsolationForestModel(uid, isolationTrees, numSamples).setParent(this))

    if ($(contamination) > 0.0) {
      val scores = isolationForestModel
        .transform(df)
        .map(row => OutlierScore(row.getAs[Double]($(scoreCol))))
        .cache()
      val outlierScoreThreshold = scores
        .stat.approxQuantile("score", Array(1 - $(contamination)), $(contaminationError))
        .head
      isolationForestModel.setOutlierScoreThreshold(outlierScoreThreshold)

      val observedContamination = scores
        .map(outlierScore => if(outlierScore.score >= outlierScoreThreshold) 1.0 else 0.0)
        .reduce(_ + _) / scores.count()
      val verificationError = if (${contaminationError} == 0.0) {
        $(contamination) * 0.01
      } else {
        ${contaminationError}
      }
```

```
      if (math.abs(observedContamination - $(contamination)) > verificationError) {

      }
    } else {
    }

    isolationForestModel
  }

  override def transformSchema(schema: StructType): StructType = {

    require(schema.fieldNames.contains($(featuresCol)),
      s"Input column ${$(featuresCol)} does not exist.")
    require(schema($(featuresCol)).dataType == VectorType,
      s"Input column ${$(featuresCol)} is not of required type ${VectorType}")

    val outputFields = schema.fields

    StructType(outputFields)
  }
}

case object IsolationForest extends DefaultParamsReadable[IsolationForest] {

  override def load(path: String): IsolationForest = super.load(path)
}
```

IsolationTree.scala

```
package ua.kpi.master.isolationforest

import ua.kpi.master.isolationforest.Nodes.{ExternalNode, InternalNode, Node}
import ua.kpi.master.isolationforest.Utils.DataPoint
import org.apache.spark.internal.Logging

import scala.annotation.tailrec
import scala.collection.mutable.ListBuffer
```

```
import scala.util.Random



private[isolationforest] class IsolationTree(val node: Node) extends Serializable {

  import IsolationTree._

  private[isolationforest] def calculatePathLength(dataInstance: DataPoint): Float =
    pathLength(dataInstance, node)
}

private[isolationforest] case object IsolationTree extends Logging {



  def fit(data: Array[DataPoint], randomSeed: Long, featureIndices: Array[Int]): IsolationTree = {

    def log2(x: Double): Double = math.log10(x) / math.log10(2.0)
    val heightLimit = math.ceil(log2(data.length.toDouble)).toInt

    new IsolationTree(
      generateIsolationTree(
        data,
        heightLimit,
        new Random(randomSeed),
        featureIndices))
  }

  def generateIsolationTree(
    data: Array[DataPoint],
    heightLimit: Int,
    randomState: Random,
    featureIndices: Array[Int]): Node = {

    def generateIsolationTreeInternal(
      data: Array[DataPoint],
```

```
currentTreeHeight: Int,
heightLimit: Int,
randomState: Random,
featureIndices: Array[Int]): Node = {



def getFeatureToSplit(data: Array[DataPoint]): (Int, Double) = {


  val availableFeatures = featureIndices.to[ListBuffer]
  var foundFeature = false
  var featureIndex = -1
  var featureSplitValue = 0.0


  while (!foundFeature && availableFeatures.nonEmpty) {
    val featureIndexTrial = availableFeatures
      .remove(randomState.nextInt(availableFeatures.length))
    val featureValues = data.map(x => x.features(featureIndexTrial))
    val minFeatureValue = featureValues.min.toDouble
    val maxFeatureValue = featureValues.max.toDouble


    if (minFeatureValue != maxFeatureValue) {
      foundFeature = true
      featureIndex = featureIndexTrial
      featureSplitValue = ((maxFeatureValue - minFeatureValue) * randomState.nextDouble
        + minFeatureValue)
    }
  }
  (featureIndex, featureSplitValue)
}


val (featureIndex, featureSplitValue) = getFeatureToSplit(data)
val numInstances = data.length


if (featureIndex == -1 || currentTreeHeight >= heightLimit || numInstances <= 1)
  ExternalNode(numInstances)
```

```
      else {
        val dataLeft = data.filter(x => x.features(featureIndex) < featureSplitValue)
        val dataRight = data.filter(x => x.features(featureIndex) >= featureSplitValue)

        InternalNode(
          generateIsolationTreeInternal(dataLeft, currentTreeHeight + 1, heightLimit, randomState,
featureIndices),
          generateIsolationTreeInternal(dataRight, currentTreeHeight + 1, heightLimit, randomState,
featureIndices),
          featureIndex,
          featureSplitValue)
      }
    }

    generateIsolationTreeInternal(data, 0, heightLimit, randomState, featureIndices)
  }


  def pathLength(dataInstance: DataPoint, node: Node): Float = {


    def pathLengthInternal(dataInstance: DataPoint, node: Node, currentPathLength: Float): Float = {

      node match {
        case externalNode: ExternalNode =>
          currentPathLength + Utils.avgPathLength(externalNode.numInstances)
        case internalNode: InternalNode =>
          val splitAttribute = internalNode.splitAttribute
          val splitValue = internalNode.splitValue
          if (dataInstance.features(splitAttribute) < splitValue) {
            pathLengthInternal(dataInstance, internalNode.leftChild, currentPathLength + 1)
          } else {
            pathLengthInternal(dataInstance, internalNode.rightChild, currentPathLength + 1)
          }
      }
    }
```

```
    pathLengthInternal(dataInstance, node, 0)
  }
}
class WriteAheadLogBackedBlockRDD[T: ClassTag](
    sc: SparkContext,
    @transient private val _blockIds: Array[BlockId],
    @transient val walRecordHandles: Array[WriteAheadLogRecordHandle],
    @transient private val isBlockIdValid: Array[Boolean] = Array.empty,
    storeInBlockManager: Boolean = false,
    storageLevel: StorageLevel = StorageLevel.MEMORY_ONLY_SER)
  extends BlockRDD[T](sc, _blockIds) {

  require(
    _blockIds.length == walRecordHandles.length,
    s"Number of block Ids (${_blockIds.length}) must be " +
      s" same as number of WAL record handles (${walRecordHandles.length})")

  require(
    isBlockIdValid.isEmpty || isBlockIdValid.length == _blockIds.length,
    s"Number of elements in isBlockIdValid (${isBlockIdValid.length}) must be " +
      s" same as number of block Ids (${_blockIds.length})")

  @transient private val hadoopConfig = sc.hadoopConfiguration
  private val broadcastedHadoopConf = new SerializableConfiguration(hadoopConfig)

  override def isValid(): Boolean = true

  override def getPartitions: Array[Partition] = {
    assertValid()
    Array.tabulate(_blockIds.length) { i =>
      val isValid = if (isBlockIdValid.length == 0) true else isBlockIdValid(i)
      new WriteAheadLogBackedBlockRDDPartition(i, _blockIds(i), isValid, walRecordHandles(i))
    }
  }
```

```
override def compute(split: Partition, context: TaskContext): Iterator[T] = {
  assertValid()
  val hadoopConf = broadcastedHadoopConf.value
  val blockManager = SparkEnv.get.blockManager
  val serializerManager = SparkEnv.get.serializerManager
  val partition = split.asInstanceOf[WriteAheadLogBackedBlockRDDPartition]
  val blockId = partition.blockId

  def getBlockFromBlockManager(): Option[Iterator[T]] = {
    blockManager.get[T](blockId).map(_.data.asInstanceOf[Iterator[T]])
  }

  def getBlockFromWriteAheadLog(): Iterator[T] = {
    var dataRead: ByteBuffer = null
    var writeAheadLog: WriteAheadLog = null
    try {

      val nonExistentDirectory = new File(
        System.getProperty("java.io.tmpdir"), UUID.randomUUID().toString).toURI.toString
      writeAheadLog = WriteAheadLogUtils.createLogForReceiver(
        SparkEnv.get.conf, nonExistentDirectory, hadoopConf)
      dataRead = writeAheadLog.read(partition.walRecordHandle)
    } catch {
      case NonFatal(e) =>
        throw new SparkException(
          s"Could not read data from write ahead log record ${partition.walRecordHandle}", e)
    } finally {
      if (writeAheadLog != null) {
        writeAheadLog.close()
        writeAheadLog = null
      }
    }
    if (dataRead == null) {
      throw new SparkException(
```

```
          s"Could not read data from write ahead log record ${partition.walRecordHandle}, " +
            s"read returned null")
      }
      logInfo(s"Read partition data of $this from write ahead log, record handle " +
        partition.walRecordHandle)
      if (storeInBlockManager) {
        blockManager.putBytes(blockId, new ChunkedByteBuffer(dataRead.duplicate()), storageLevel)
        logDebug(s"Stored partition data of $this into block manager with level $storageLevel")
        dataRead.rewind()
      }
      serializerManager
        .dataDeserializeStream(
          blockId,
          new ChunkedByteBuffer(dataRead).toInputStream())(elementClassTag)
        .asInstanceOf[Iterator[T]]
    }

    if (partition.isBlockIdValid) {
      getBlockFromBlockManager().getOrElse { getBlockFromWriteAheadLog() }
    } else {
      getBlockFromWriteAheadLog()
    }
  }

  override def getPreferredLocations(split: Partition): Seq[String] = {
    val partition = split.asInstanceOf[WriteAheadLogBackedBlockRDDPartition]
    val blockLocations = if (partition.isBlockIdValid) {
      getBlockIdLocations().get(partition.blockId)
    } else {
      None
    }

    blockLocations.getOrElse {
      partition.walRecordHandle match {
        case fileSegment: FileBasedWriteAheadLogSegment =>
```

```
          try {
            HdfsUtils.getFileSegmentLocations(
              fileSegment.path, fileSegment.offset, fileSegment.length, hadoopConfig)
          } catch {
            case NonFatal(e) =>
              logError("Error getting WAL file segment locations", e)
              Seq.empty
          }
        case _ =>
          Seq.empty
      }
    }
  }
}
```

CSV.java

```
package ua.kpi.master;

import java.util.stream.Collectors;
import java.util.stream.Stream;

public class CSV {
  public static String convertToCSV(String[] data) {
    return Stream.of(data)
        .map(CSV::escapeSpecialCharacters)
        .collect(Collectors.joining(","));
  }

  private static String escapeSpecialCharacters(String data) {
    String escapedData = data.replaceAll("\\R", " ");
    if (data.contains(",") || data.contains("\"") || data.contains("'")) {
      data = data.replace("\"", "\"\"");
      escapedData = "\"" + data + "\"";
    }
    return escapedData;
```

```
  }
}
```

Executor.java

```
package ua.kpi.master;


import ua.kpi.master.isolationforest.IsolationForest;
import ua.kpi.master.isolationforest.IsolationForestModel;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.ml.feature.VectorAssembler;
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics;
import org.apache.spark.sql.Column;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;


import java.util.Arrays;


public class Executor {
  public static void main(String[] args) throws Exception {
    final String filePath = args.length > 0 ? args[0] : "";
    int numOfFeatureCols = args.length > 1 ? Integer.parseInt(args[1]) : 0;


    SparkSession spark = SparkSession
        .builder()
        .master("local")
        .appName("IsolationForest")
        .getOrCreate();


    JavaSparkContext jsc = new JavaSparkContext(spark.sparkContext());


    Dataset<Row> rawData = spark
        .read()
        .format("csv")
```

```
    .option("comment", "#")
    .option("header", "false")
    .option("inferSchema", "true")
    .load(filePath);

String[] cols = rawData.columns();
String labelCol = cols[cols.length - 1];

if (numOfFeatureCols == 0) {
  numOfFeatureCols = cols.length;
}

String[] copyOfRange = Arrays.copyOfRange(cols, 0, numOfFeatureCols);

VectorAssembler assembler = new VectorAssembler()
    .setInputCols(copyOfRange)
    .setOutputCol("features");

Dataset<Row> data = assembler
    .transform(rawData)
    .select(new Column("features"), new Column(labelCol));

System.out.println(">>>>>>>>>>>>>>>>>>>>>>>");
data.show();
System.out.println("<<<<<<<<<<<<<<<<<<<<<<<");

double contamination = 0.10;
IsolationForest isolationForest = new IsolationForest();
isolationForest.setNumEstimators(1);
isolationForest.setBootstrap(false);
isolationForest.setMaxSamples(256);
isolationForest.setMaxFeatures(1.0);
isolationForest.setFeaturesCol("features");
isolationForest.setPredictionCol("predictedLabel");
isolationForest.setScoreCol("outlierScore");
```

```
    isolationForest.setContamination(contamination);
    isolationForest.setContaminationError(0.01 * contamination);
    isolationForest.setRandomSeed(1L);

    long startTime = System.nanoTime();
    IsolationForestModel isolationForestModel = isolationForest.fit(data);
    long endTime = System.nanoTime();

    long timeElapsed = endTime - startTime;

    System.out.println("Execution time in nanoseconds  : " + timeElapsed);

    System.out.println("Execution time in milliseconds : " +
        timeElapsed / 1000000);

    System.out.println("TRESH" + isolationForestModel.getOutlierScoreThreshold());

    Dataset<Row> transform = isolationForestModel.transform(data);

    System.out.println("============================");
    transform.show();
    System.out.println("============================");

    isolationForestModel.write().overwrite().save("./tmp/result");

    IsolationForestModel forestModel = IsolationForestModel.load("./tmp/result");

    BinaryClassificationMetrics metrics = new BinaryClassificationMetrics(transform.select("outlierScore",
"predictedLabel"));

    System.out.println(metrics.areaUnderROC());
    spark.stop();
  }

}
```

ДОДАТОК Б – ГРАФІЧНИЙ МАТЕРІАЛ

ПЛАКАТ 1 ІНФОРМАЦІЙНІ ПОТОКИ ПРОЦЕСУ МАШИННОГО НАВЧАННЯ

ПЛАКАТ 2 СХЕМА МАШИННОГО НАВЧАННЯ З ВИКОРИСТАННЯМ ПОТОКОВОЇ ОБРОБКИ ДАНИХ